School of Computing
FACULTY OF ENGINEERING

Social Proofs with Blockchain Consensus

Pengyuan Huang

Submitted in accordance with the requirements for the degree of MSc. Advanced Computer Science (Artificial Intelligence)

2019/2020

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Deliverable 1	Report	SSO & VLE (27/08/20)
Deliverable 2	URL	Supervisor & Assessor (27/08/20)
Deliverable 3	Developer Setup Guide	Supervisor & Assessor (27/08/20)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

Summary

Existing social media platform, most of which hold the privacy of citizens, and using centralized machine learning algorithms can post fake information to manipulate citizens. Citizens often feel unable to truly and effectively influence public affairs.

This project aims to use blockchain smart contracts to store all information into the block to implement a decentralized application (DApp), ensure that information data are immutability and traceable, implement social proof in blockchain and consensus network.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Dr Evangelos Pournaras. He gave me valuable advice during our weekly meetings to help me complete this project.

I would also like to thank my assessor, Dr Raymond Kwan, who provided helpful feedback both in the progress meeting and on the interim report, all of which contributed to the successful completion of this project.

Finally, I would like to thank my family and my friends for their continuous support during the project and even throughout the university. Without them, the project would not have been successful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Aim

The main aim of this project is to use the blockchain consensus to explore social proofs of witness presence, and implement it as an interactive Android application through the blockchain smart contracts.

1.2 Project Objectives

The project has the following objectives:

- Develop blockchain smart contracts and deploy on the Ethereum network.
- Develop Android Application that call smart contracts by scanning barcode to verify the social proof.
- Integrate the system into Smart Agora Platform.
- Test the entire system to evaluate its practical application.

1.3 Project Justification

The initial application of blockchain technology was electronic currency, known as Bitcoin, but with further research and innovation, the new generation of Ethereum blockchain is available now. At the meantime, through smart contracts, the Ethereum blockchain is no longer limited to digital currency transactions, and has been applied in the real economy such as finance, insurance, notarization and retail. I chose this project because the research on blockchain related projects will be of great help to my further studies [1].

1.4 Deliverables

At the end of the project, the following items will be uploaded:

- A MSc project report in PDF format detailing the research and development of this software.
- A Github repository link for storing all source code of the software.
- Appropriate documentation for the installation, configuration, instructions for setting up the project.

1.5 Ethical, legal and social issues

1.5.1 Ethical

This project strives to avoid all ethical issues, and no real personal data is stored in the software. All the data of this project are stored in the Ethereum test network, which has security guarantee. Every transaction made can be viewed on the website.

1.5.2 Legal

The project uses an open source library and is also completely open source. It will fully comply with the provisions of the open source agreement.

1.5.3 Social

The social issues of the project is mainly to help solve the problem that citizens are being used by social media to manipulate personal data and even privacy by using machine learning algorithms to make correct decisions for public affairs. It uses the blockchain to provide witness presence to help citizens participate in decision-making.

1.6 Dissertation Structure

This report is organized into the following chapters that present different aspects of the project:

Chapter 1 - Introduction provides an introduction to the project.

Chapter 2 - Background Research states in detail the background related to the project and the techniques to be used.

Chapter 3 - Software Requirements and System Design describes the requirements of the software and explain the overall design of the software without giving any details about its implementation.

Chapter 4 - Software Implementation describes the implementation process in detail of the project.

Chapter 5 - Software Testing and Evaluation will outline the methodology for testing software, the outcome of the tests as well as the evaluation of the project.

Chapter 6 - Conclusions and Future Work is conclusions of this research project and outline any future work to improve and develop the work on the project.

Chapter 2

Background Research

The background research of this project covers several fields such as witness presence and social proof, Blockchain, Ethereum and smart contracts, analyzing why using them is a better choice by detailing their characteristics.

2.1 Literature Review

Existing social media can collect citizens' personal data and even privacy, isolate citizens through centralized machine learning algorithms, and use false information to manipulate citizens. It is difficult for citizens to contribute to public affairs without being affected by social media platforms. In order to address these challenges, a new approach to blockchain consensus is introduced by proving witness presence to help citizens participate in decision-making [2]. One of the cores of witness presence is social proof, which verifies the situation awareness required for collective decision-making.

2.1.1 Witness Presence

The presence of witness is mostly a statutory construct borne out of the 19th-century law relating to wills. However, witness presence also gives citizens the right to make decision. They can directly influence smart cities decisions through intervention and testimony instead of being a realistic passive spectator.

2.1.2 Social Proofs

Some blockchain approaches combine location-based and social-based proofs to verify the presence of witness. However, in some cases of unpredictable movement and weak signal, network-based location proof may be unavailable or perform poorly. In these scenarios, social proofs is a better choice to verify witness presence.

Social proof is a term proposed by Robert Cialdini [3] in his 1984 book, *Influence*. It describes a social phenomenon wherein people imitate the behaviour of others under specific circumstances. However, it refers to one of proof of witness presence in this project. It is an approach of verifying the presence of witness by proving that citizens have localized nearby the point of interest to make decisions, including Contextual QR Codes [4], which are based on different points of interest use corresponding physical social proofs.

2.2 Blockchain

The blockchain is known to be a decentralized public ledger technology for all transactions [5], which was first proposed by Satoshi Nakamoto in the 2008 paper "*Bitcoin: A Peer-to-Peer*

Electronic Cash System" [6]. It is a new application model that combines innovative technologies such as distributed networks, cryptographic algorithms, and consensus mechanisms. Blockchian-based systems do not require third-party to audit it but maintain an immutable and forgivable distributed ledger through cryptography. It uses a consensus mechanism to maintain transactions in the decentralized node system and ensure a unified ledger, while mainly consists of the following basic concepts:

- Transaction: Every operation on the blockchain is considered a transaction, and each transaction corresponds to a unique transaction hash value.
- Block: In the blockchain network, data will be permanently recorded in the form of files, while these files called blocks. Each block is divided into two parts: block header and block body [7].
- Chain: Connects each block into a blockchain based on the timestamp sequence on the block.

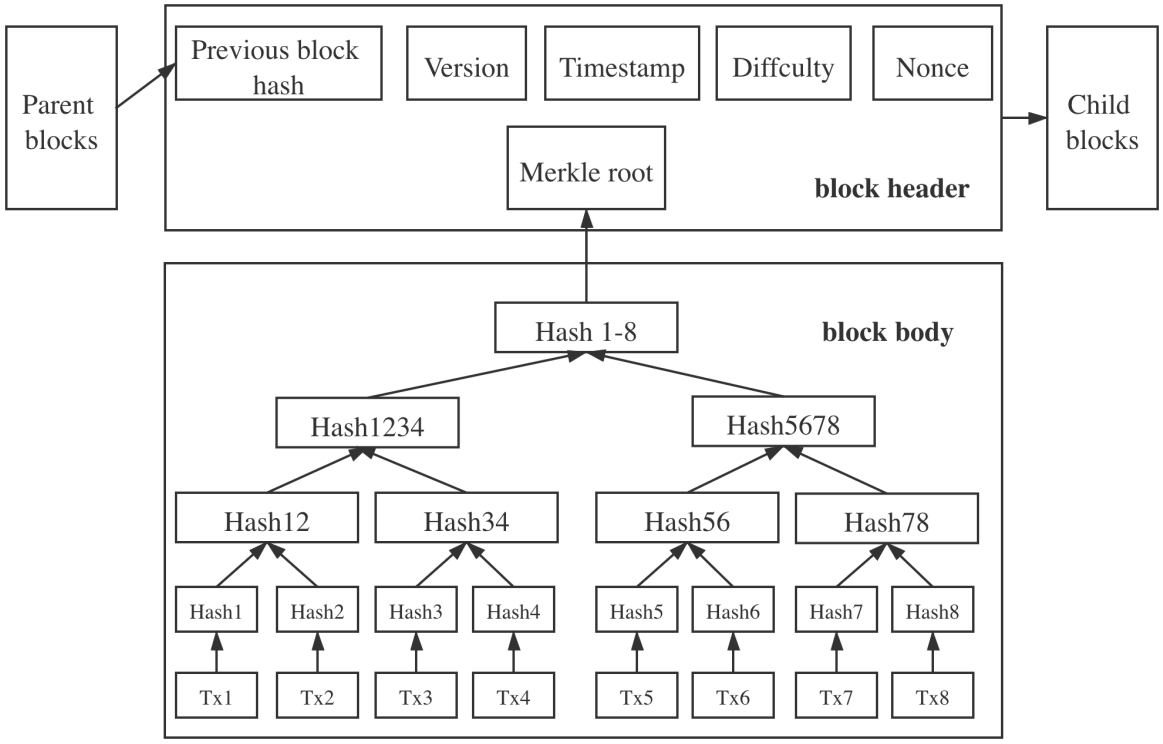


Figure 2.1: Blockchain Structure

The blockchain connects the blocks determined by the consensus mechanism according to the timestamp sequence to form a chain, as shown in Figure 2.1. Block header records the meta-information of the current block, including the previous block hash, timestamp, Merkel root hash, while the block body records the actual data information.

2.2.1 Blockchain Layered Approach

The basic technical components of each layer of the blockchain include transaction, block, consensus. The main aspects of blockchain can be divided into six layers from top to bottom:

application, contract, incentive, consensus, network and data layers, while each layers has different characteristics as shown below [5].

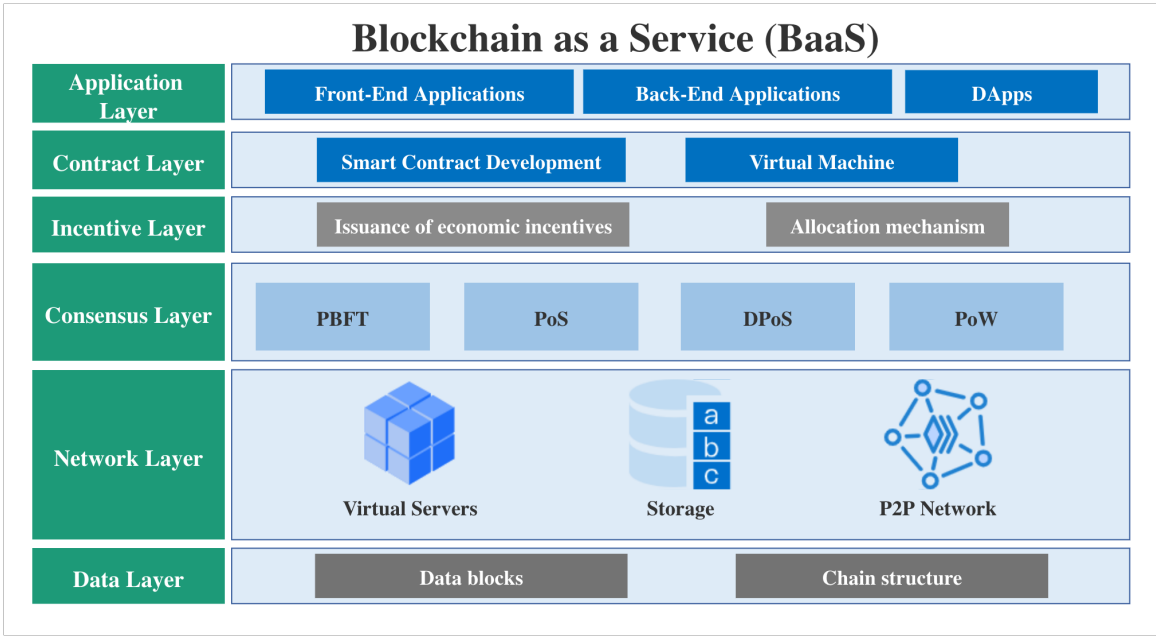


Figure 2.2: Proposed Illustration of Blockchain Layers

The top layer is the application layer, including front-end and back-end applications. The second top layer is the contract layer, including smart contracts. Generally speaking, the incentive layer refers to the reward of mining in the blockchain, and rewards some digital assets to incentivize miners to verify the transaction information to maintain the mining and keep the ledger updated [8]. The consensus layer contains many kinds of consensus algorithms, such as Proof of Work (PoW), Proof of Stake (PoS) and Delegated Proof of Stake (DPoS). The network layer refers to a P2P network based on Ethereum or Hyperledger nodes [5]. The bottom layer is the data layer, including the underlying data block of the chain structure and timestamp technology.

2.2.2 Classification of Blockchain

There are three types of blockchain technology: public blockchain; blockchain, which belongs to consortium and completely private blockchain [9].

The public blockchain is the public, fully decentralized blockchain, which means that anyone can access any node in the blockchain network. The public blockchain, as a derivative of decentralized trust, uses cryptography to ensure the immutability of data on the chain with proof of work mechanisms, and its reward mechanism incentivizes more nodes to mine. Bitcoin and Ethereum are the most famous examples of public blockchain.

Consortium blockchain is decentralized or polycentric blockchain. In the consensus process, specific nodes are selected to control the member blocks to achieve consensus in advance. For example, a blockchain system composed of N financial institutions, each institution will

participate only if 2/3 N or mode nodes pass the validation process. Also, consortium blockchain has higher security and performance requirements for the time and state.

The existence of management nodes in the private blockchain has made many people controversial about private blockchain, arguing that they defeat the original purpose of blockchain technology. However, the most significant advantage of private blockchain is encryption and auditing, so they are often used for database management with organizations and companies.

2.2.3 Consensus

A consensus mechanism is a multi-party collaborative mechanism, which can coordinate multiple participants to reach a single, mutually acceptable result and ensures that the process is hard to be deceived. In the blockchain system, the consensus mechanism aims to solve the problem of trust between nodes [10]. Initially, Bitcoin used the Proof of Work (PoW) mechanism, which mainly relies on the computing power of nodes for mining. With the improvement of blockchain technology, the consensus mechanism is continuously being innovated, people have proposed some mechanisms that do not rely excessively on computing power to make each node of the blockchain achieve network-wide consistency, such as Proof of Stake (PoS) mechanism, Delegated Proof of Stake (DPoS) mechanism, Practical Byzantine Fault Tolerance (PBFT) algorithm [11].

PoW

Proof of Work is a mechanism to prevent double-spends. It was the first consensus algorithm proposed and is still the dominant algorithm. In the blockchain system, this mechanism is used to confirm transactions and generate new blocks on the chain. The main advantages of PoW are the anti-DoS attacks defense and less impact on the possibility of mining. The main disadvantages are enormous expenditures, and a 51% attack.

PoS

Proof of Stake is another blockchain consensus algorithm, which is to make distributed nodes in the blockchain to reach consensus. It is often applied with PoW. While the overall process is consistent with PoW, the approach of reaching the final goal is entirely different.

In the PoS mechanism, the production of blocks can converge on the consensus between the validators and network nodes occur much faster than its PoW counterpart. Therefore, the PoS network has higher performance in terms of network transmission and transactions. On the contrary, the disadvantage of PoS is that network can be attacked and destroyed.

DPoS

DPoS is a consensus algorithm that can be used for voting and election. It uses a professional-operated network server to ensure the blockchain network has better security and performance. This algorithm has three requirements for the system as follows:

- Randomly specify the order of producers.

- Blocks produced out of order are invalid.
- Shuffle the cards once every cycle, disrupting the original order.

Compared with PoW and PoS, the advantages of this mechanism is low power consumption and a shorter period of consensus. The disadvantage is that it is easy to cause a blockchain fork, and it is hard to deal with bad nodes.

PBFT

Practical Byzantine Fault Tolerance is a consensus algorithm introduced by Barbara Liskov and Miguel Castro [12] in the late 1990s. PBFT can work efficiently in in asynchronous systems bacause it is optimized for low overhead time. It was designed to help solve many problems related to Byzantine Fault Tolerance solutions.

The advantages of PBFT are energy efficiency, transaction finality and low reward variance. Stakeholders ensure the safety and stability of the node. The limitations of PBFT is that the PBFT consensus model cannot work efficiently when there are many nodes in the distributed network [13].

2.3 Blockchain 1.0 – Bitcoin

The representative technology of Blockchain 1.0 is a virtual currency led by Bitcoin, which represents the application of virtual currency, including its payment, circulation and other functions of virtual currency. The main feature of blockchain 1.0 is the decentralized digital currency transaction payment, which aims to achieve the decentralization of the currency and means of payment. Compared to traditional means of payment, the blockchain network can achieve peer-to-peer transactions, and do not rely on large financial institutions.

Bitcoin is the most typical representative of blockchain 1.0. It outlines a grand blueprint, the future of the currency is no longer dependent on the release of central banks, but the globalization of curreny unification.

Nevertheless, blockchian 1.0 only meets the needs of virtual currency, while it cannot be spread to other non-financial industries because the scripting system supporting smart contracts is not perfect.

2.4 Blockchain 2.0 – Ethereum

Blockchain 2.0 is based on the cryptographic digital currency technology to develop the function of smart contracts to achieve the blockchain in addition to the ability to act as a peer-to-peer cash transaction system other broader functions. The biggest breakthrough is the ability to implement simple application development on the blockchain network, namely DApp (Decentralized Application) [14].

Ethereum is a representative platform of blockchain 2.0, which can be used to deploy smart contracts.

2.4.1 Architecture

The overall architecture of Ethereum is shown in Figure 2.3 below, which is mainly divided into three layers: Underlying Layer, Core Technology Layer, Top Layer. The underlying layer guarantees the stable operation of the Ethereum blockchain system, including essential services such as P2P network, LevelDB database and cryptography algorithms. The core technology layer is the core components of Ethereum, including consensus algorithm and Ethereum Virtual Machine and other core components. With the blockchain as the main body and consensus algorithm as the assistance, the EVM is used to run smart contracts. The top-level layer is based on Ethereum platform, including API interface, smart contracts and decentralized applications and other services. The DApp sends transactions to smart contracts by the Web3 framework. The interaction between contracts and the blockchain is called by RPC.

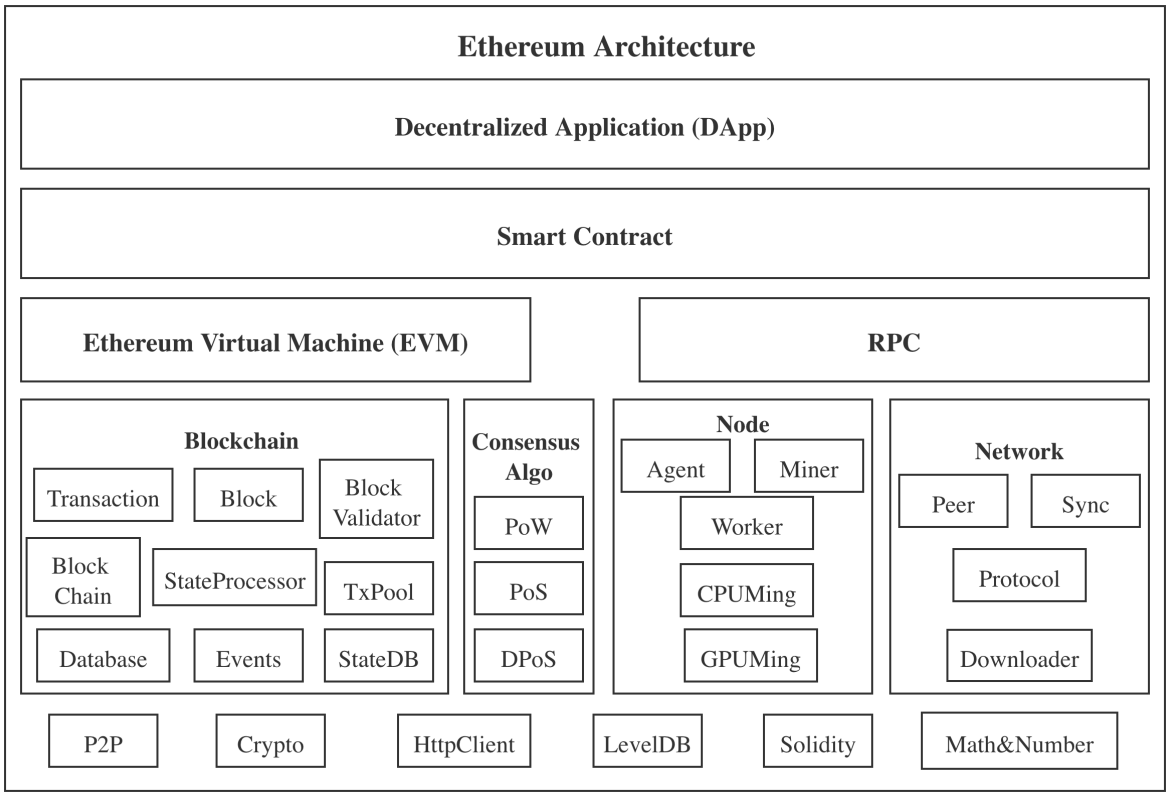


Figure 2.3: Ethereum Structure

2.4.2 Account

In the Ethereum blockchain, users are required to have an account in order to make transactions. There are two different types of accounts in the Ethereum, one is crypto-currency account, known as an externally owned account, and the other is controlled by smart contracts code [15].

2.4.3 Gas

Gas is a unit in EVM that measures how much it costs to transact or execute a contract, and how much it takes to perform an act of workload. It was designed to avoid code execution all

the time, and is a safe way for both miners and users to avoid the error code.

2.5 Blockchain 3.0

According to the industry at present, blockchain 3.0 has not formed an industry-wide consensus, mainly to improve the performance and expansion of the blockchain 2.0 [16].

Although there is no recognized 3.0 representative project, there have been some breakthroughs in scalability, transaction, consensus mechanism. According to the current technological progress, blockchain 3.0 can be widely used in large-scale industry applications, such as the fields of government, education, medical care, technology. This means that the blockchain platform is becoming a service environment like traditional internet technology, which can realize from the bottom layer to user-friendly application to better support.

Regrading the blockchain 3.0 scenario, the following applications may be available in the future:

- Automated procurement
- Intelligent IoT applications
- Supply chain automation management

2.6 Smart Contracts

Smart contracts were first proposed by Nick Szabo [17] in 1994, which can automatically monitor, execute and implement legal agreements.

In Ethereum, contract actions are manipulated by the contract code, and the account storage always retains the state of the contract. Since the contract is driven by events (transactions), it is indeed to get the sender address, the receiver address, the gas values and other attributes, and then encapsulate the transaction-specific information into events. If the state machine trigger condition (one or more) is met, then the contract is automatically executed according to the predefined data.

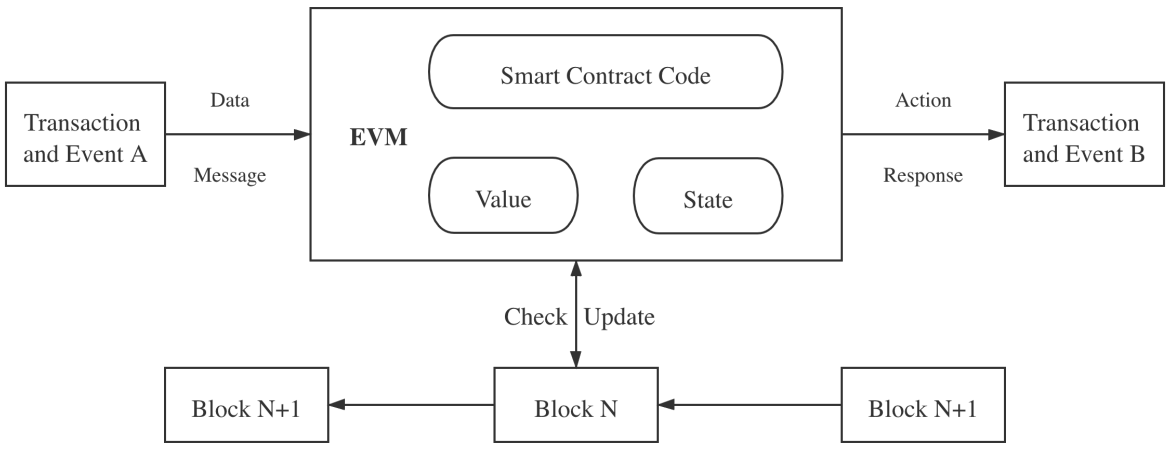


Figure 2.4: Smart Contract Model

The model of smart contacts is illustrated in Figure 2.4 above. When the transaction and event A are sent to the contract, the contract analyzes and processes event A and generates a different transaction and event B according to the trigger conditions, and records the status and value of the contracts on the blockchain.

Ethereum has four special languages that can be used to develop smart contracts:

- Solidity, inspired by JavaScript
- Serpent, inspired by Python
- Mutan, inspired by Go
- LLL, inspired by Lisp

These four programming languages are all designed from the bottom for contract-oriented programming, but from the current development [18], Solidity is the well-deserved language of choice for Ethereum smart contract development.

2.6.1 Solidity

The Origins of Solidity

Computer scientist Gavin Wood originally initially proposed the concept of the Solidity language [19]. A brief history is given below:

- August 2014: The Solidity language is proposed by Gavin Wood.
- Solidity is adopted as a language by Monax, a rival platform.
- Solidity is officially released.

What can Do with Solidity

The programming language provides developers with the skills to create their own decentralized apps (DApps). Just as applications in the Apple app store are built to run on iOS, DApps on Ethereum are built to run on Solidity. Many applications of Solidity include the following aspects:

- Voting: In the real world, the voting process is prone to many forms of fraud, such as the manipulation of data and voting machines. These problems can be solved by using contracts for voting. Solidity can be used to plan the code, and with proper implementation, the process of voting will become smooth, transparent and automatic.
- Crowdfunding: If done through contracts, crowdfunding can solve various problems related to commissions for third-party organizations [20].

Also, Solidity has an effort on proof of witness presence, avoid incorrect implementations which could cause consensus problems.

2.6.2 Truffle Suite

The evolution of the Truffle Suite is making DApp development look over more attractive. It has three components: Truffle, Ganache, Drizzle [21].

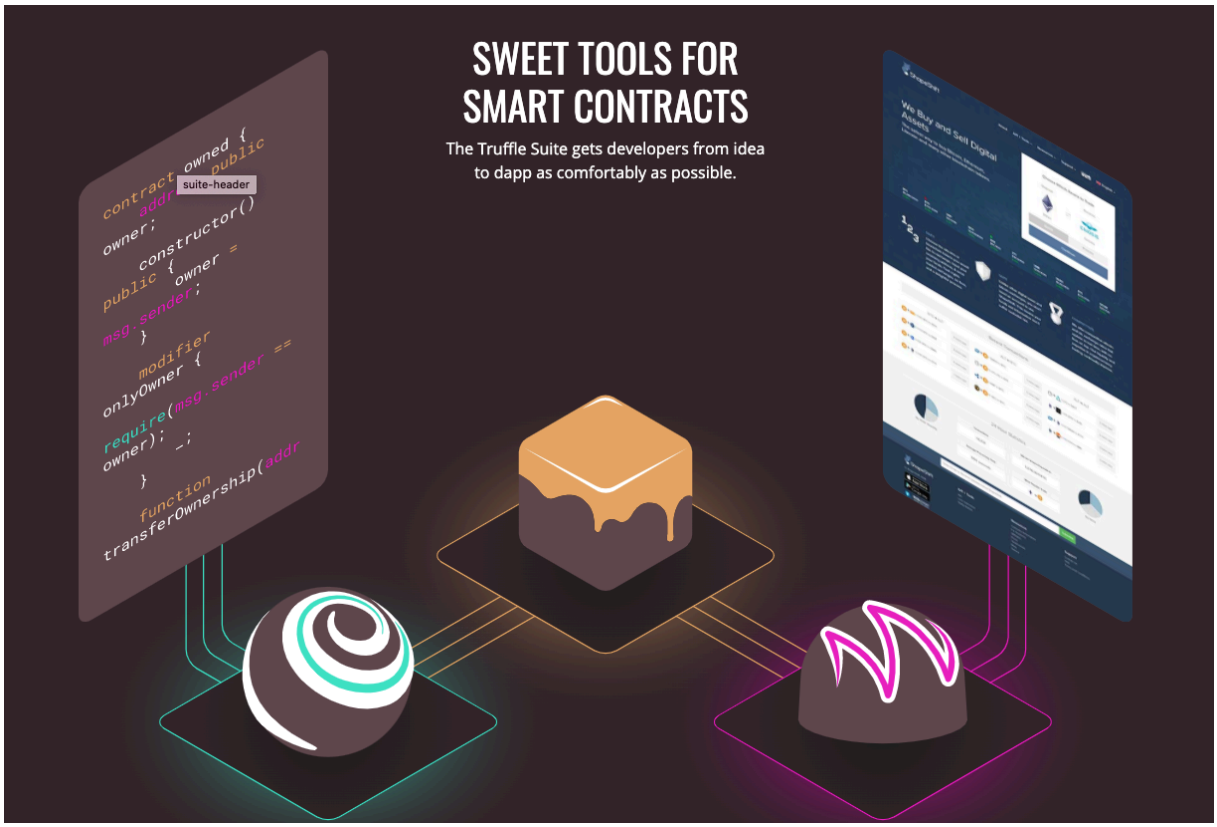


Figure 2.5: Truffle Suite

Truffle

Truffle is a development environment, test framework for Ethereum blockchains [22], which aimsing to make it easier to develop smart contracts. With Truffle, developer gets:

- Automated contract testing.
- Deployment of public and private networks with migration framework.
- Use the console to operate the contract directly.

Hence, Truffle is a very handy tool that makes development easier. The smart contracts of this project will be developed based on Truffle.

Drizzle

Drizzle is a collection of front-end libraries that make developing DApp front-ends easier. It has the following features:

- Fully reactive contract data, including data, events and transactions.
- Declarative

- Maintains access to underlying functionality. Web3 and contract's methods are untouched.

However, this project is based on Android platform which means Drizzle is not helpful for this project.

Ganache

Ganache is personal Ethereum blockchain designed for development, testing and perform other tasks without any cost, which simulates the features of the real Ethereum network [23].

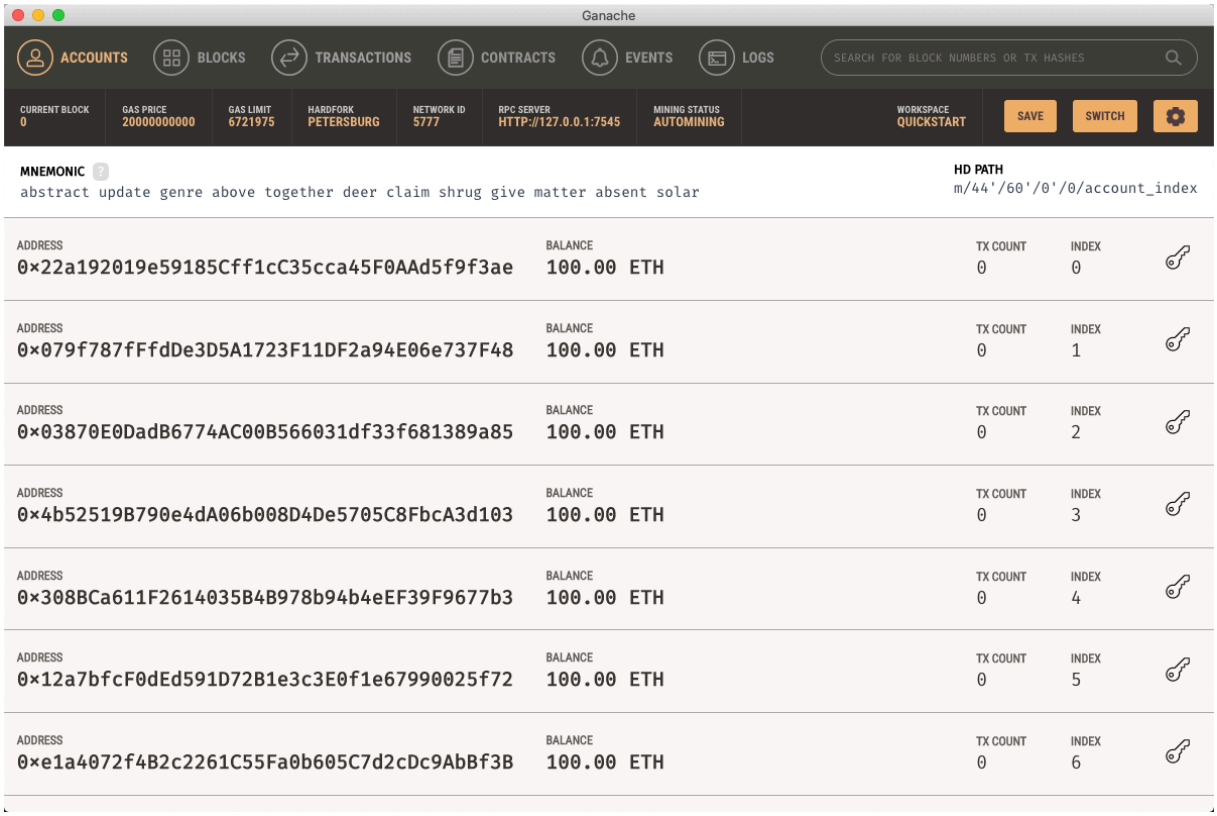


Figure 2.6: RPC Server of Ganache

Smart contracts are hard to program. Once deployed, smart contracts cannot be modified and each deployment has an associated cost. In order to debug and test smart contracts before putting into the production environment, it is therefore essential to allow developers to recreate blockchain environments locally, without increasing inconvenience of deployment costs and transaction delays.

Ganache can be used to initialize the genesis block easily, and it also prepares ten funded and unlocked accounts. However, it is designed for a local in-memory Ethereum blockchain, which is not suitable for Android applications because they should not be limited to local networks.

2.6.3 Remix

Remix is an open source web-side IDE for the development of Solidity smart contracts [24]. It provide basic functions such as compiling, deploying to a local or test network, and executing

contracts. It does not need to be installed and can be launched directly in any browser.

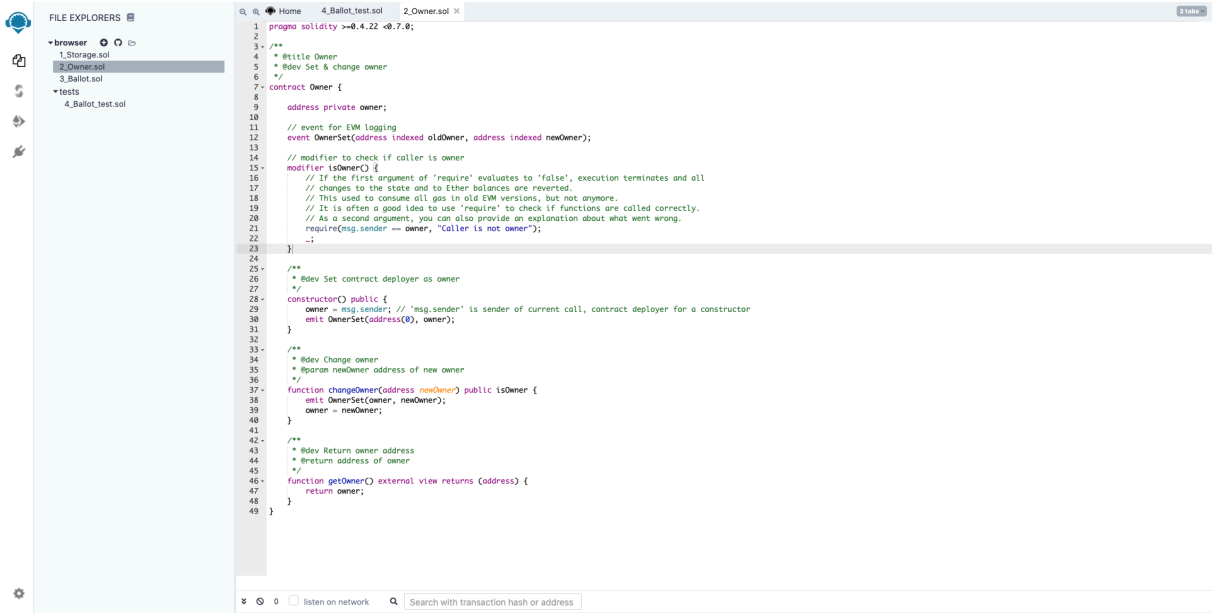


Figure 2.7: Remix

2.6.4 MetaMask

MetaMask provides a simple way for browsers to interact with smart contracts [25]. It is an extension of Chrome or Firefox that allows running smart contracts on the browser. Through MetaMask, developers can deploy smart contracts in the Ethereum main network, or in locally created blockchain networks. This project will deploy smart contracts in the test networks with MetaMask.

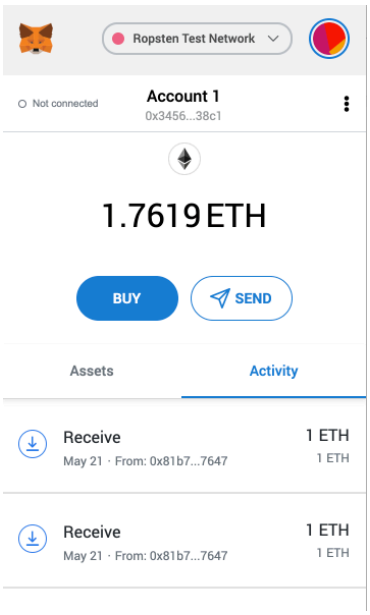


Figure 2.8: MetaMask Detail

2.6.5 Infura

Infura is a utility that can lower the barrier for using Ethereum blockchain, which can connect applications to the public Ethereum network.

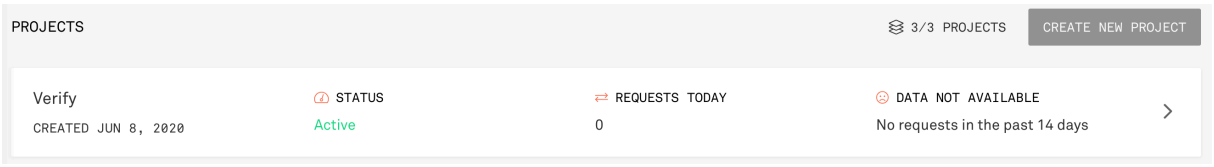


Figure 2.9: The Project in Infura

2.7 Choice of methods

2.7.1 Platforms

By analyzing the characteristics of the three different versions of the blockchain, it can be concluded that the blockchain 1.0 is mainly suitable for digital currencies in the financial field, but does not support the development of any application based on the operating system. Blockchain 2.0 adds smart contracts to the previous foundation, becoming a programmable blockchain in the true sense. This stage supports Turing complete scripting language and provides the necessary infrastructure for developers to develop applicatoins. Blockchain 3.0 is an idealized vision of people for the future virtual digital currency economy. At present, there is no representative project and it is not mature enough.

In summary, Ethereum based on blockchain 2.0 is the most suitable technology for this project, because blockchain 1.0 does not support the development of non-financial fields, while blockchain 3.0 is still in inmature state. The mature technology and community of Ethereum can help implement this project.

2.7.2 Tools

Similarly, there are two ways to do Solidity programming, one is to use Truffle and the other is to use Remix. They both provide a development environment for Solidity smart contracts that can be easily compiled, deployed and executed.

However, Remix cannot be version-controlled, tested, or used with other development tools and is better suited for entry-level developers.

Therefore, the final combination is based on the Ethereum blockchain network, using Solidity to develop smart contracts in the development environment provided by Truffle, and using MetaMask and Infura to deploy and connect to the public Ethereum test network.

Chapter 3

Software Requirements and System Design

3.1 Software Requirements

The basic software requirement of the project is based on smart contracts development and deployment on the Ethereum public network, so that all citizens in smart cities can use Android application on their smartphones to obtain location information and use physical evidence to implement social proof, making citizens not affected by the social media manipulates the use of the right to make public decisions.

Detailed requirements are divided into functional requirements and non-functional requirements.

3.1.1 Functional Requirements

The project contains the following specific functional requirements:

- Need to deploy the smart contract to the Ethereum network.
- Help users quickly obtain their own location information when they enter the application interface.
- The user can manually add a place as a point of interest by adding a marker in Google Maps.
- The user can cancel a certain point of interest on the map.
- The user can click on a place on the map, choose to set it as a social proof point, and complete verification.
- The user can verify the authenticity of the information and complete the previous step of verification by scanning and identifying the QR code based on the provided test data.
- The complete verification process not only needs to consider whether the information of the physical evidence is true and valid, but also consider the user's geographic location. Sometimes it is necessary to use secondary verification to prove that the user has arrived at a certain place at a certain time.
- The user can directly add verification, delete, change the test data and generate the QR code.
- The project manager can add test data and generate QR codes on the contract side.
- The project manager can modify test data and generate QR codes on the contract side.
- The project manager can delete test data and generate QR codes on the contract side.

All the above operations will interact with the smart contract deployed on the Ethereum network through the Android application, and will be verified by comparing with the data in the blockchain.

3.1.2 Non-Functional Requirements

Non-functional requirements of this project have the following key elements: maintainability, performance, security and deployability.

Maintainability

This project is only the realization of a part of the paper "*Proof of Witness Presence: Blockchain Consensus for Augmented Democracy in Smart Cities*" [2], so it is normal to constantly change the requirements simultaneously, which means it needs to have good maintainability.

Security

Security is a core part of non-functional requirements because the original intention of this project is to prevent citizens from being overly manipulated by social media and the big data and machine learning algorithms of giant companies, protecting users' privacy and other personal data is the core requirement of this project. In terms of security, it is necessary to use encryption algorithms multiple times and store all data in a decentralized blockchain network to prevent data leakage and tampering.

Deployability

In terms of deployability, the current project only requires smart contracts to be deployed in the Ethereum test network due to the cost of test network is smaller. However, as the project matures and the number of users increases, the performance of the test network cannot support load balancing. Therefore, this project needs to be able to migrate to the official Ethereum network or self-built network.

3.2 System Design

3.2.1 Architecture Overview

The architecture of the system is divided into five layers. The first layer is the blockchain layer, which communicates with the network layer, and also as the decentralized database of the system. This layer stores data and information about the user's interaction with the system, and includes technologies such as P2P networks, consensus mechanisms, encryption algorithms and timestamp.

The second layer is the Ethereum Virtual Machine. The EVM is similar to the Java Virtual Machine, which is the operating environment of the Ethereum smart contracts.

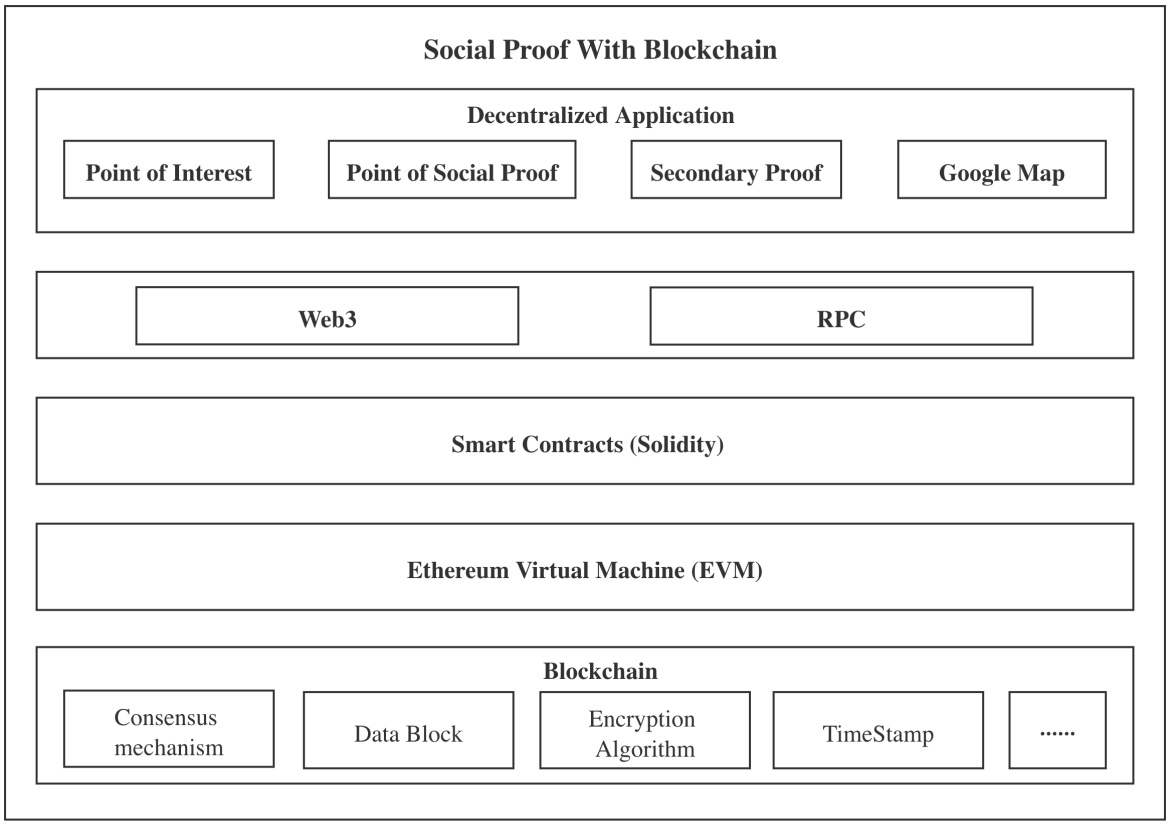


Figure 3.1: System Structure of the Project

The third layer is smart contract written in the Solidity language, which is also the business logic layer of the system. By compiling the Solidity code into EVM bytecode, and then, deploying it to the Ethereum network to execute.

The fourth layer is the Web3 library. It encapsulates the JSON RPC and provides a way to communicate with any Ethereum nodes that expose the RPC interface to communicate.

The fifth layer is the application layer which is the social proof system. It interacts with the user through the Android interface, where the user can add points of interest or point of social proof to verify that the information is valid and that the user information meets the requirements. All the underlying data is stored on the Ethereum blockchain to ensure of immutability of the data.

3.2.2 Use Case

Proof of witness presence can help citizens make more informed decisions in smart cities. The working paper "*On cycling risk and discomfort: urban safety mapping and bike route recommendations*" [26] examines the risks of cycling in cities. According to this paper, a logical use case would be to design a collective movement to get citizens' opinions, provide valuable suggestions for urban planners to improve infrastructure, and even increase the risk awareness of cyclists. For instance, citizens rate the safety of cycling in different locations in cities and give more reasonable opinions, such as new bike lanes. Citizens need to prove their position, are in or have reached the point of interest of the designated decision-making, in order to be

more likely to obtain more productive and valuable decisions [2].

The citizens can interact with the system in the following approach. As shown in Figure X, in the city map, the system designers or city planners can set several points of interest in advance, and each interest points contains some logically related questions. When a collective movement is initiated, citizens can answer or evaluate questions via their smartphones only when they reach the point of interest. In order to achieve the incentive, citizens can earn crypto-currency rewards by answering questions [2].

The use case diagram as shown below:

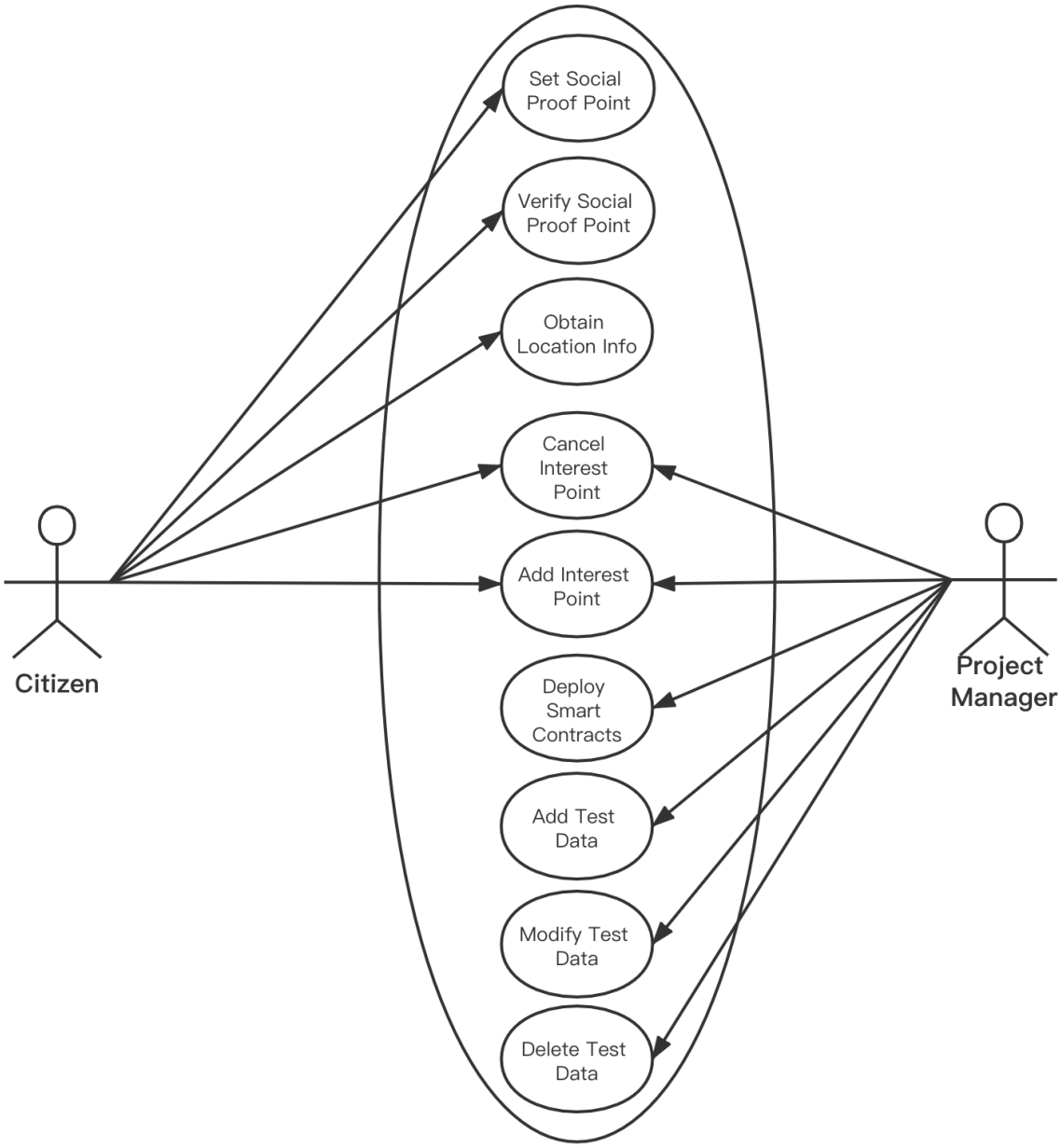


Figure 3.2: Use Case Diagram

3.2.3 Six Different Scenarios

This project is available for all location in smart cities. There are too many location types in the Google Map API, which are too specific and not representative. Therefore, this project summarizes six generic scenarios as followings according to all the location types in Google Map which mean users can use different physical evidence to prove situation awareness according to different locations [27].

- Transport Ticket (station, airport, etc.)
- Receipt (restaurant, mall, etc.)
- Id Card (school, accommodation, etc.)
- Ticket (park, gallery, etc.)
- Prescription (hospital, pharmacy, etc.)
- Certificate (company, insurance agency, etc.)

For example, citizens can use a train ticket to proof that this user has arrived at a train station within a certain time. Because when citizens buy a ticket online or offline, this record will be committed on the blockchain smart contracts. Hence, anyone can verify whether it is valid through an entity, such as barcode or QR Code on the ticket. Additionally, each entity has an attribute of time that can be used to prove that this user arrived at the location within a period, which means the entity is not valid at all times and has an expiration date.

3.2.4 Secondary Social Proof

Since many public places, such as parks and art galleries, are open to the public free of charge, it may not be possible to use the admission ticket to prove that the user has been to this place within a period of time. Therefore, it is necessary to use the location around a point of interest for secondary social proof.

When direct social proofs are unavailable, secondary social proof is needed to find several locations around a point of interest that the user wants to prove that he or she has been to in a period. Then, the system assigns weights based on the distance between the points of social proof and a point of interest from near to far.

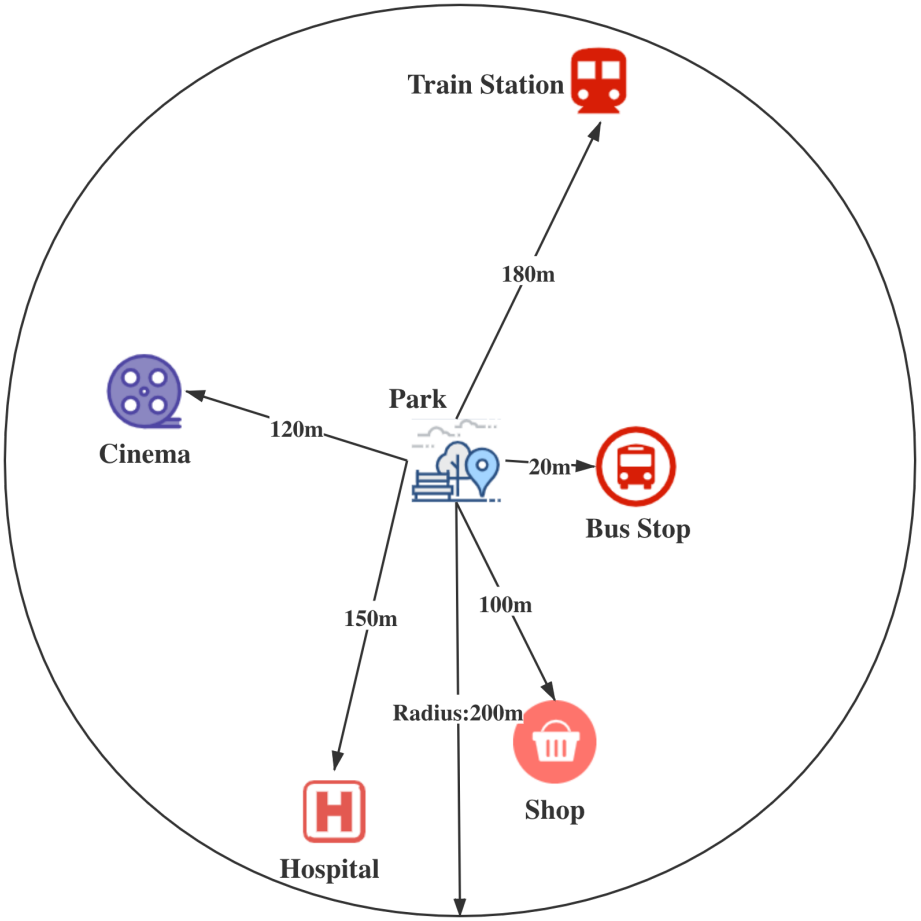


Figure 3.3: Secondary Social Proof

As Figure 3.3 shown above, setting a park as a point of interest can be given by an oracle [27]. In addition, there are some locations such as a bus stop, a shop, a cinema, a hospital and a train station (from near to far) within the scope of this oracle.

If a user cannot prove that he or she has been to this point of interest directly, then the user can choose to use secondary social proof. The user can find nearby locations that are within the range of this oracle, such as cinema and bus stop, and prove that the user has been to those locations in a period time.

It is not scientifically to assign weights in order, such the nearest weight should be 0.5. Because it is possible that the nearest one is 150 meters apart and do not reach the weight of 0.5 that should be persuasive. The weights should be assigned according to distance with a specific formula to calculate weights:

$$weight = (range - distance)/range \tag{3.1}$$

For example, a bus stop is 20 meters away from the park, and then weight is calculated to 0.9 (because the range is 200 meters in Figure 4.2), which is a very high weight, meaning that the

user can prove that he or she has been to this point of interest (the park) by proving that the user has come to the bus stop convincingly.

In fact, direct social proof can also be regarded as secondary social proof. Because if the user can prove that he or she has been to this point of interest directly, then the distance should be 0 and the weight should be 1. This means that if the weight is 1, then the secondary social proof can be regarded as a direct social proof, which can directly prove that the user has been to the point of interest.

Similarly, because the weights are cumulative, if the total weight of each point of social proof after accumulation is greater than or equal to 1, it means that social proof is successful. On the contrary, it proves a failure.

3.3 Project Management

3.3.1 Version Control

The use of Git allows anyone related to this project to track any changes to the source code during the project development process and can provide reasonable opinions related to project development and management. Therefore, the code developed by this project is hosted in a public repository in Github. Additionally, using a public repository is to ensure that anyone can use the project under the open source license or contribute to the project through pull request.

3.3.2 License

Anyone can use this project under the premise of following the license of the project since it will be open-source. The project is licensed under the GPL-2.0 License [], which has some key features as follows:

- The source code of the copied or modified or distributed program must also be open sourced under the GPL-2.0.
- No warranty.
- It can be used for business but must be open source.

Chapter 4

Software Implementation

4.1 Implement and Deploy Smart Contracts

After researching about related work such as programming language and deployment environment, the next step is to formally develop and deploy smart contracts.

4.1.1 Initialize the Project with Truffle

We use Truffle to initialize the project, its directory structure is shown in Figure 4.1:

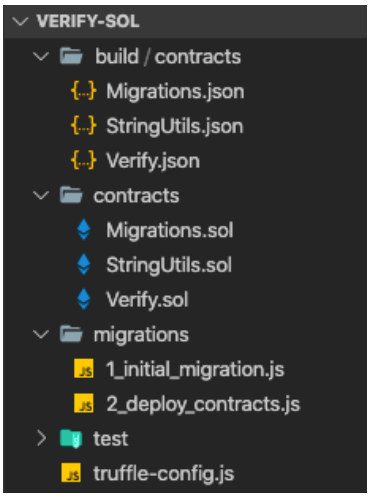


Figure 4.1: Truffle Project Directory

The specific project directory structure description is shown below:

- build: There will be stored JSON files that automatically generated after compiling.
- contracts: There will be stored all Solidity (.sol files). Any required smart contracts, libraries or interfaces will be added here at compile time.
- migrations: There will be stored Javascript (.js files) for deploying smart contracts.
- test: There will be stored test script files.
- truffle-config.js: Here configs which networks to use, gas usages and other variables.

4.1.2 Module Implementation: Transport Ticket as an Example

The smart contracts contain many different modules, such as the seven different scenarios mentioned before. Here is an example of the transport ticket.

Transport ticket has many necessary attributes, such as the name of owner and price. Thus, the structure design is as follows:

Attribute	Type
id	uint256
ownerName	string
from	string
to	string
price	uint256
timeDeparture	string
timeArrival	string

Table 4.1: Transport Ticket Structure

In addition, there are the structural design of the remaining 6 scenarios:

Attribute	Type
id	uint256
place	string
time	string
total	uint256
paid	uint256
change	uint256
payType	string

Table 4.2: Receipt Structure

Attribute	Type
id	uint256
patientName	string
age	uint256
addr	string
date	string
tel	uint256
mobile	uint256
doctorName	string

Table 4.3: Ticket Structure

Attribute	Type
id	uint256
cardNo	string
cardId	string
ownerName	string
expireDate	string
organization	string
identity	string

Table 4.4: Id Card Structure

Attribute	Type
id	uint256
no	string
typeOf	string
name	string
place	uint256
givenAt	string
time	string

Table 4.5: Certificate Structure

Attribute	Type
id	uint256
ticketNo	string
place	string
data	string

Table 4.6: Prescription Ticket Structure

Attribute	Type
id	uint256
name	string

Table 4.7: Point of Interest Structure

After the definition of structure, it is needed to define the key-value mapping of the structure as the primary key. Also, set the primary key “id” to auto increment so that the code is as follows:

```
mapping (uint256 => TransportTicket) transportTickets;
mapping (bytes32 => uint256) hashToTransportTicketId;
uint256 transportTicketIndex = 0;
```

The transport ticket module has some essential functions, such as add, query, check. The developer of this system needs to use the add function to store a large amount of test data into the block in advance, otherwise, it cannot be verified because this project is a simulation process that does not have real data. After adding the test data, anyone can use the query or check function to verify the authenticity of the data.

It is worth noting that not all data is added to the block separately, but all the data is spliced into a string, then performs the Hash operation to store the obtained hash value into the block. The query or check function is to determine whether the hash value is the same and whether it meets the timestamp requirements.

4.1.3 Deploy Smart Contracts

After registering for MetaMask, developers can receive test coins in the Ropsten test network. Developers need to get the mnemonic belonging to this account in MetaMask, which need to be configured in truffle-config.js in the next subsection.

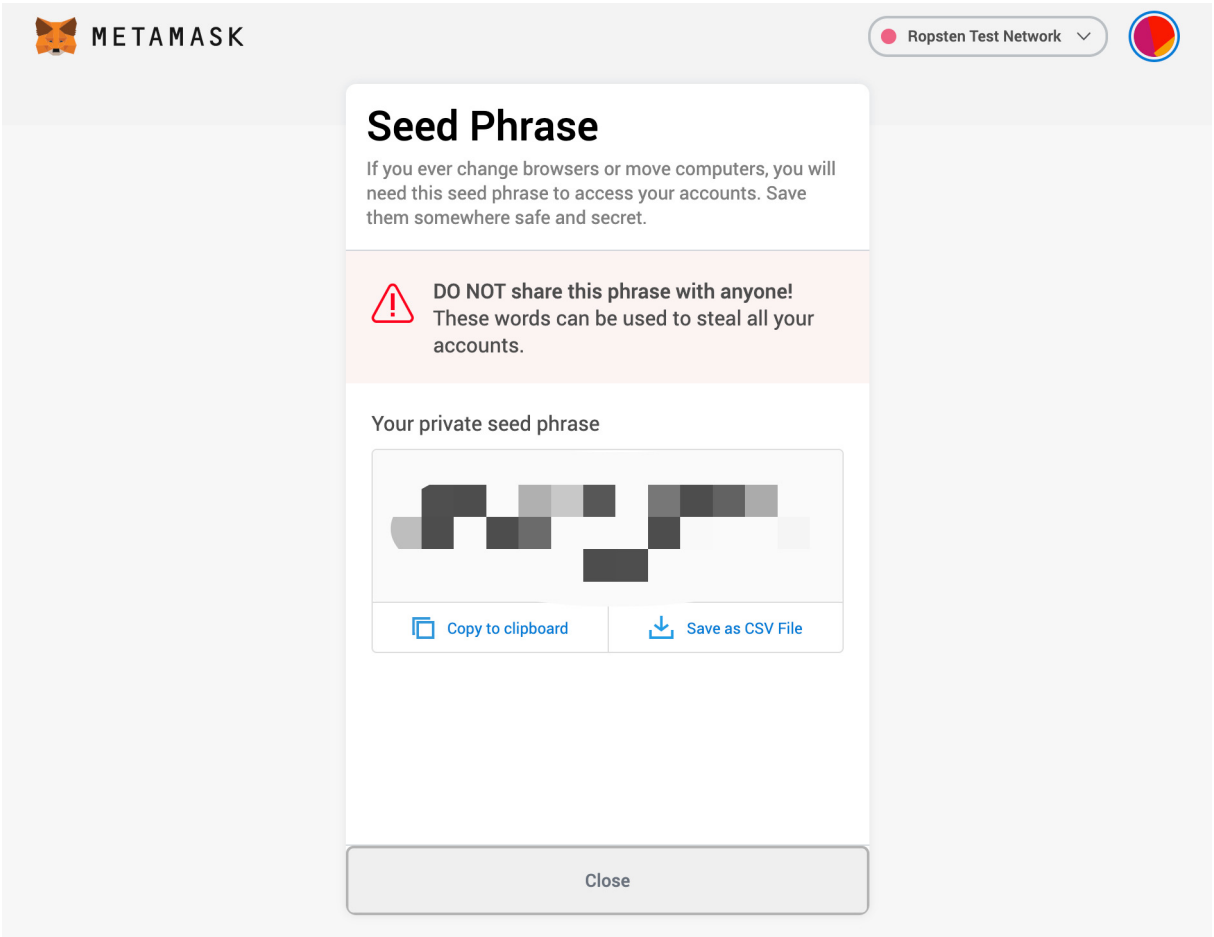


Figure 4.2: Mnemonic in MetaMask

The Ethereum client network hosted in Infura, which spans four Ethereum networks: Mainnet, Ropsten, Rinkeby, and Kovan [28]. This project will be migrated to the Ropsten test network. Create a new project on Infura's website, and then automatically generate an API key, a secret key and a network address, which need to be configured in truffle-config.js in next subsection.

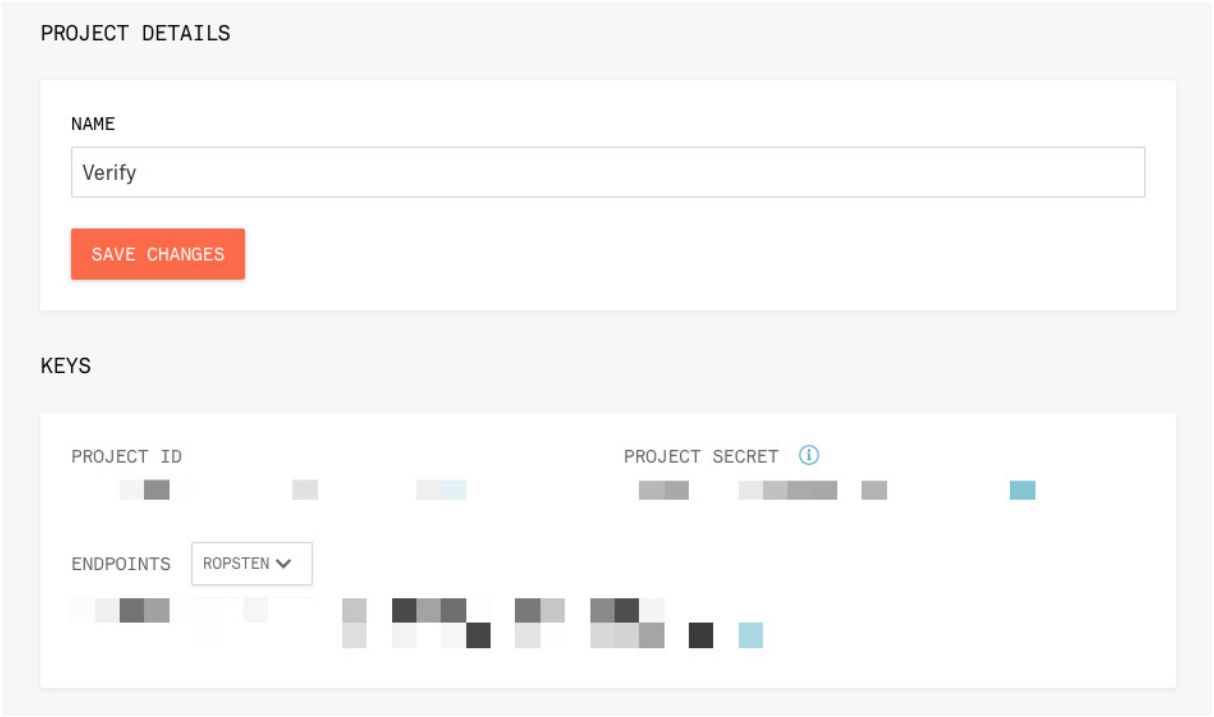


Figure 4.3: Infura Settings

Edit the truffle-config.js file to enable HDWalletProvider. This plugin can help smart contracts connect to account wallets. Need to download the HDWalletProvider plugin which is a separate npm package by using the following command:

```
npm install @truffle/hdwallet-provider
```

The next step is to change the configuration in truffle-config.js based on the key information obtained by Infura and mnemonic of MetaMask:

```
const HDWalletProvider = require('@truffle/hdwallet-provider');

// 12 seed words in MetaMask -> Settings -> Security & Privacy -> Reveal Seed
    Words
const mnemonic = "orange apple banana ...";

module.exports = {
    networks: {
        reposten: {
            provider: new HDWalletProvider(mnemonic, "https://ropsten.infura.io/
    v3/<INFURA_PROJECT_ID>"),
            network_id: 3
        }
    }
};
```

Next, compile the contracts, and then use the following command to migrate the contracts to the Ropsten network:

```
truffle migrate --reset --network ropsten # --reset optional
```

Then developers can get all the output information in the console as follows, including the contract address. Anyone can view all the information about the contract, including every transaction, at https://ropsten.etherscan.io/tx/contract address.

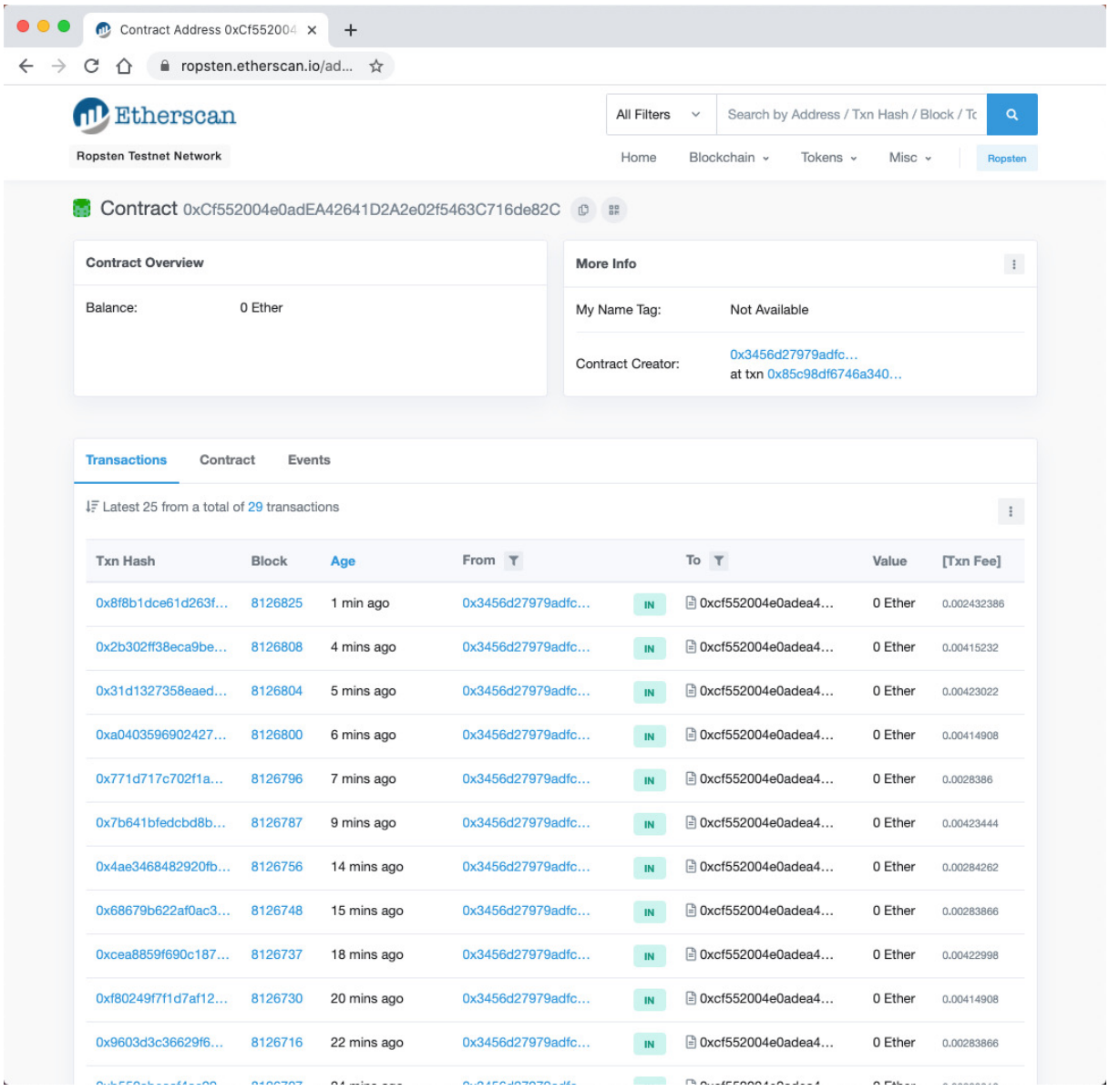


Figure 4.4: Contract Address of the Project

4.2 Android Application Implementation

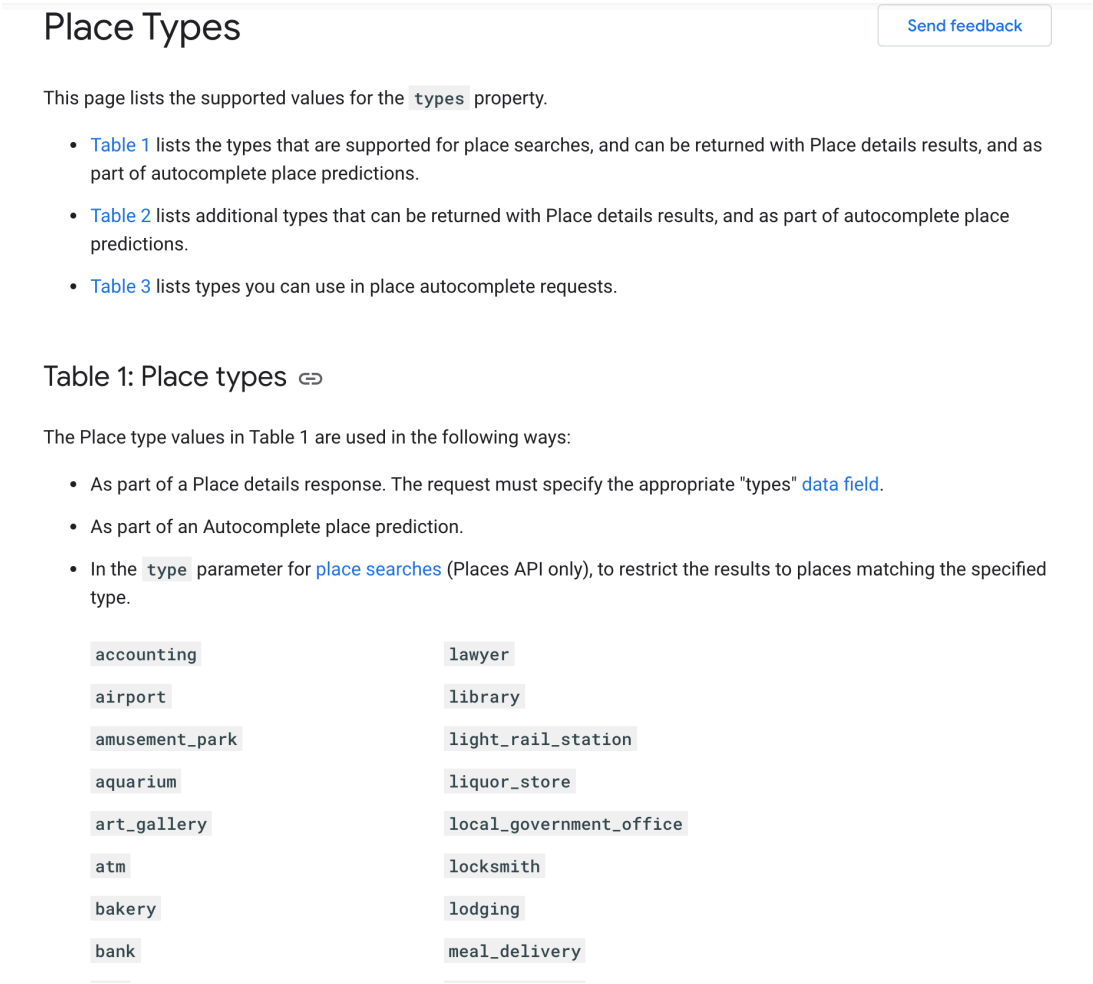
4.2.1 Google Map

Google Maps is a map service developed by Google, which provides different transportation services. On September 23, 2008, Google installed the Google Maps application for its first

Android smartphone. Map Marker can help this DApp to set points of interest and points of social proof on the Google Map.

Points of interest and points of social proof are two relatively uncommon and different concepts. The point of interest, which can often be abbreviated as POI, is a landmark or scenic spot on an electronic map to mark the government part, commercial institution, tourist attraction, transportation facility and other places represented by the spot [29]. A point of interest must contain information such as name, type, longitude, and latitude, which will be displayed on the map [29]. The API provided by Google Maps has the concept of point of interest.

The point of social proof, on the other hand, is a new term created by this project. It means that citizens can prove where they have been. After the point of interest is created, citizens participating in this project need to prove that they have been present somewhere near the point of interest over a period of time. They can happen to appear at a point of interest, or they can appear in the vicinity near a point of interest. Regardless, locations that can be proven by citizens that they have been for a period of time are called the point of social proof.



Place Types

Send feedback

This page lists the supported values for the `types` property.

- Table 1 lists the types that are supported for place searches, and can be returned with Place details results, and as part of autocomplete place predictions.
- Table 2 lists additional types that can be returned with Place details results, and as part of autocomplete place predictions.
- Table 3 lists types you can use in place autocomplete requests.

Table 1: Place types

The Place type values in Table 1 are used in the following ways:

- As part of a Place details response. The request must specify the appropriate "types" data field.
- As part of an Autocomplete place prediction.
- In the `type` parameter for place searches (Places API only), to restrict the results to places matching the specified type.

`accounting`	`lawyer`
`airport`	`library`
`amusement_park`	`light_rail_station`
`aquarium`	`liquor_store`
`art_gallery`	`local_government_office`
`atm`	`locksmith`
`bakery`	`lodging`
`bank`	`meal_delivery`

Figure 4.5: Google Maps SDK for Android – Place API

Therefore, this project can use the Google Maps SDK for Android to add markers to locations on the map, set this location as a point of interest or a point of social proof. The marker

represents a single location on the map and info window of marker can provide additional context for this marker.

Additionally, with the Maps SDK for Android, this application can get the distance between two markers and all information about the location, including name and type. These key factors help implement this project.

4.2.2 Two-Dimensional Barcode

Two-dimensional barcode, especially QR Codes, are used to quickly encode and decode data [4].

There is a wide variety of existing 2D barcode, and the two most famous 2D barcode standards are DataMatrix and QR Code [4], while QR Codes are the most common and popular in the world.

Technology

QR Code is a two-dimensional barcode that introduced by a Japanese company Denso-Wave in 1994 [30], which can be easily interpreted by scanning equipment.

Without the smart device, it is impossible for humans to decode two-dimensional barcodes, while the machine can quickly recognize the code. Citizens can view the information on the device only by scanning the barcode through a smart device, such as an application in their smartphone. Hence, in this project, the most secure approach to store information is through QR codes.

Development

There are many open-source projects about two-dimensional barcode in Github, and ZXing provided by Google [31] is the most famous one. It is an image processing library implemented in Java, while this project will integrate the ZXing library.

Figure **??** is the two pre-stored test data in this project, two locations in Leeds, UK. One is represented by barcode, and the other is represented by QR Code. Citizens can scan barcodes similar to the following to obtain information.

(a) 2D Barcode of Central Village

(b) QR Code of Leeds Station

Figure 4.6: 2D Barcode and QR Code

4.2.3 Implement Android Application

After researching about related works such as Google Map and barcode, the next step is to develop Android applications.

Import Google Maps SDK

First, import Google Maps SDK for Android into this application, get an API key and enable the necessary APIs, add the API key to this application.

Next, create a map object with markers and automatically locate the user's current location after users enter the application. There may already be some points of interest (blue markers) on the map [27]. Citizens can choose a location on the map to click and then click on the info window of this location and choose to set it as a point of interest or a point of social proof. It is noted that any record of data changes will be committed to the Ethereum blockchain by calling smart contracts functions, which ensures that the data is immutability and anyone is able to verify the authenticity of the data.

When the location is set as a point of interest, a blue marker will be on the map and it can be given by a red circle (an oracle).

When citizens set the location as a point of social proof, the application will jump to the corresponding verification interface, depending on the type of location citizens click on. If the verification is successful, there will be a purple marker on the map to represent the point of social proof. Whether the location is set as a point of interest or a point of social proof, the record is synced to the Ethereum blockchain by smart contracts.

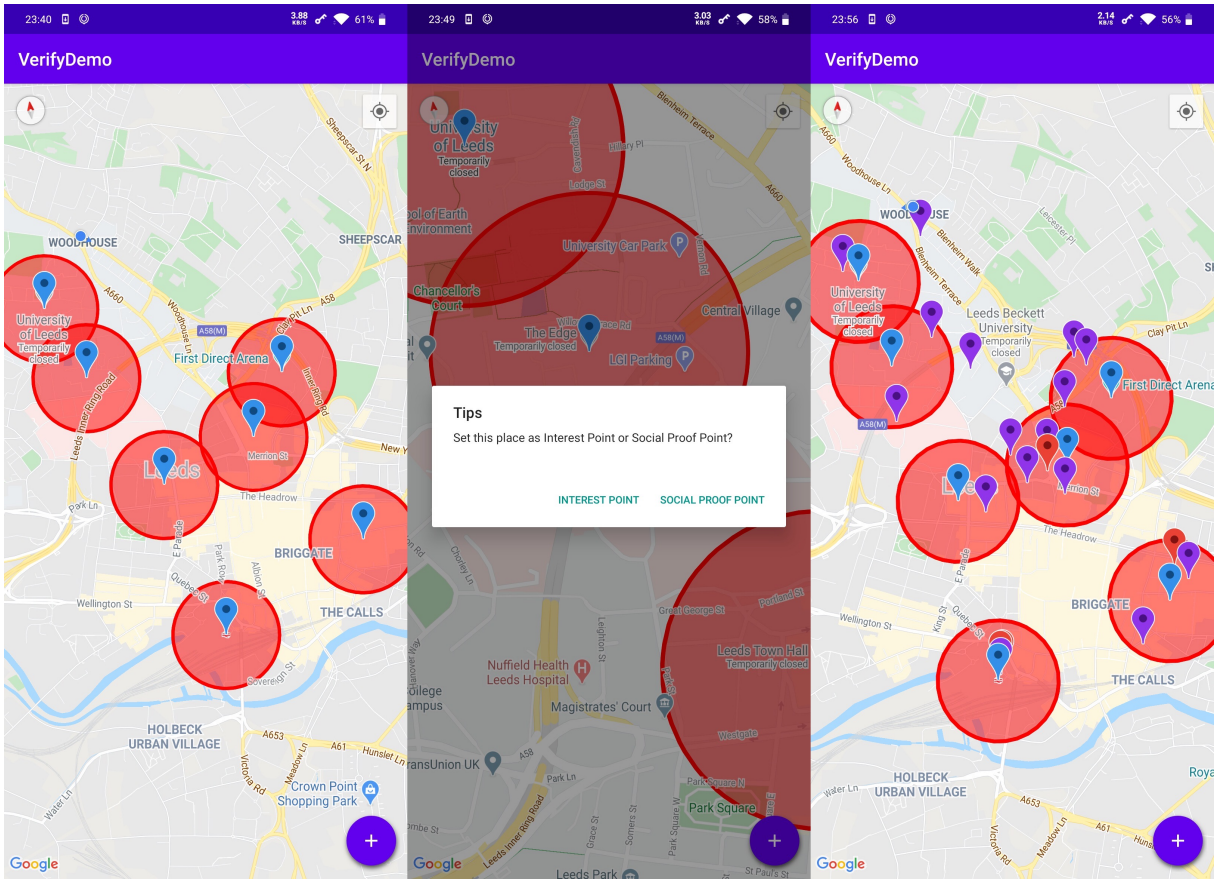


Figure 4.7: Map Interface in the Application

Social Proof by Barcode

As mentioned above, the verification process of setting point of social proof is one of the core verifications of this system. It is implemented by scanning whether the barcode data is valid.

In the Google Maps API, as many as 96 place types are provided and classified into six different scenarios in Section 3.2.3. For example, if the user chooses a train station in the map to set it as a point of social proof, then this application will jump to the verification interface of “Transport Ticket”, because the train station needs to use the entity of the train ticket to complete the verification. Similarly, if the user chooses a hospital in the map and sets it as a point of social proof, the interface will jump to the verification interface of “Prescription”, and the user needs to complete the verification according to the corresponding prescription or medical certificate information. The following uses a train ticket example to illustrate the entire verification process.

When a user purchases a train ticket online or offline, this record will be committed to the Ethereum blockchain through smart contracts. The electronic train ticket or physical train ticket received by the user will have a barcode containing all information about this train ticket. After the user enters the application’s verification transport ticket interface, clicks the “SCAN BARCODE” button to scan the barcode on the train ticket, the application will parse the data and fill the data into the form. The user can first check whether the information on the form is accurate, and then click “CHECK” button, the application will start the

verification process.

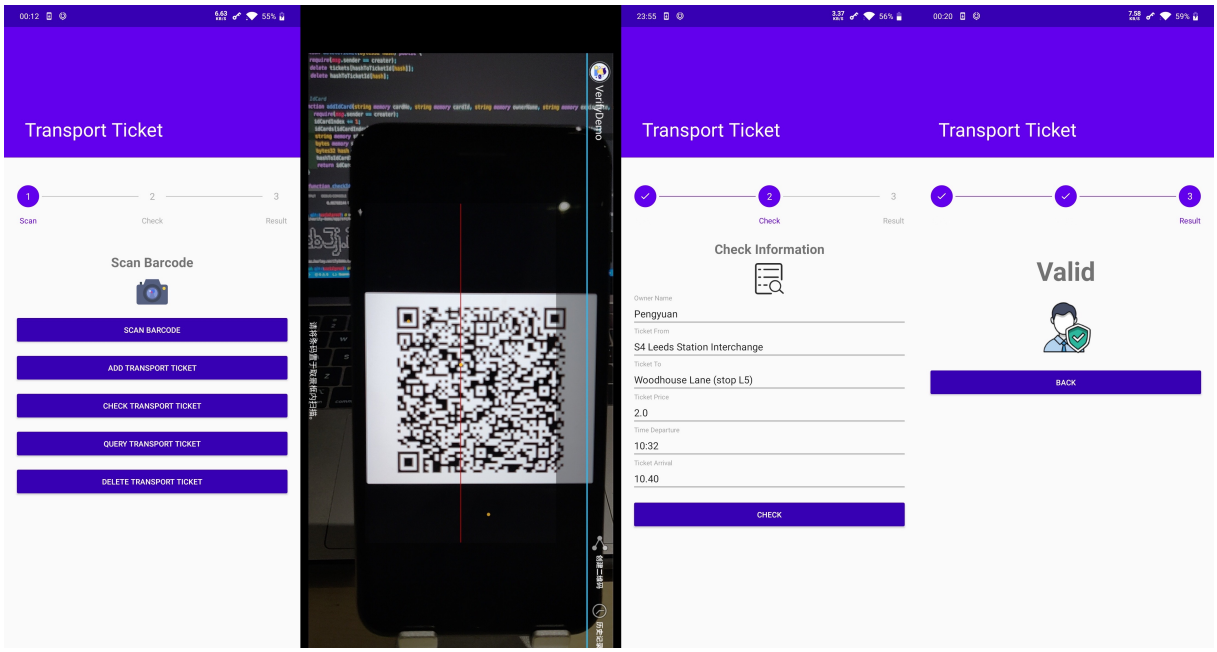


Figure 4.8: The Process of Social Proof by Scanning Barcode

The verification process mainly needs to be verified in two parts. The first is whether all the information of the train ticket is the same as the information stored in the Ethereum blockchain, which ensures the authenticity of the information through the immutability of the blockchain.

The verification method is to splice all the data obtained by scanning 2D barcode, and then perform a hash operation (SHA-256) to obtain a fixed-length hash value. In the smart contract, we have stored many test data in advance. Similarly, the entire data is merged into a string and then hashed to obtain a hash value and stored in the block. This system will compare whether the two hashes are the same. If the two hashes are equal, it is valid. According to the characteristics of the hash algorithm in Section 2.2.6, two texts with the same information are encrypted by the hash algorithm and get identical hash value. Even if one character in the two texts is different, the encrypted hash value of both texts is messy, and there is no correlation between the two hash values. Thus, this method ensures the security of the information.

Second, whether the location of this train ticket, including the departure and arrival locations, is the same as the location to be verified by the user on the map. Otherwise, even if the information of the train ticket is valid, the location of this train ticket is different from the location to be verified by the user, which means false verification.

Verify Social Proof Meets the Requirements

Verifying whether the points of social proof meets the requirements of a particular point of interest is another core verification of this system. This verification process is also divided into two parts.

The first part is the “time” attribute of the entity. The train ticket of the previous section is

also explained as an example. The time of a train ticket includes departure time and arrival time. It is meaningless for users to use an expired train ticket to prove that they have been to this train station because this project aims to implement social proofs in a time limit even if it is not real-time. This project needs to prove that the user has arrived at the location within a recent period, so the expired information is useless.

Admittedly, this time limit can be set dynamically, because different points of interest also have different time limits. For instance, policymakers will spend more time on a specific voting campaign, while it does not take too long for citizens to answer questions on their smartphones when they reach points of interest. Therefore, the time attribute of the entity needs to be within the time limit of the interest point. In this way, the point of social proof that is successfully verified is useful for the point of interest, and the weight can be assigned.

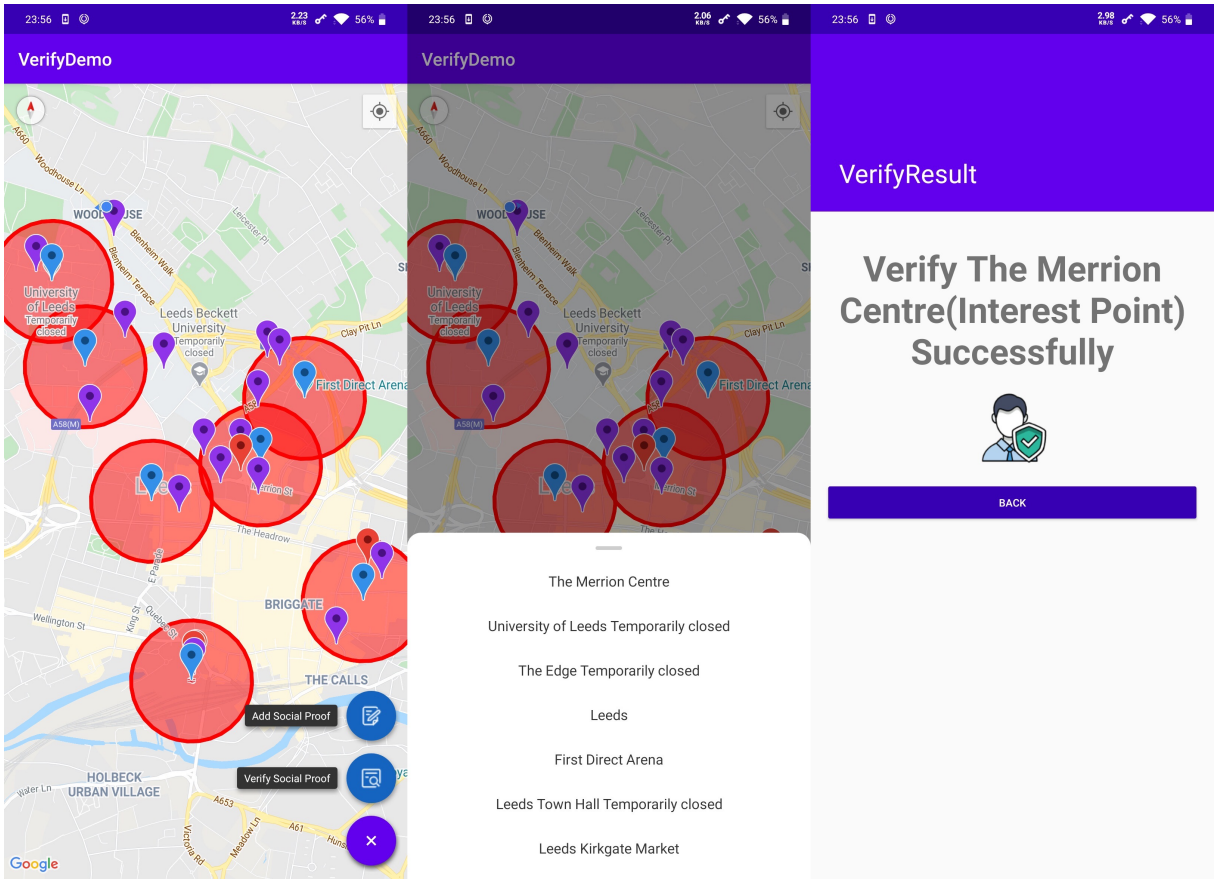


Figure 4.9: Verify Social Proof Meets the Requirements

The second part is about secondary social proof as mentioned in Section 3.2.4. For points of social proof that are within the oracle (red circle) of a certain point of interest and meet time requirements, the system is supposed to assign weights to them based on the distance between the points of social proof and a point of interest from near to far. This system calculates the weight of each point of social proof by Formula 3.1, and then adds all the weights to get the total weight. If the total weight is greater than or equal to 1, then these points of social proof can successfully prove that the user has reached this point of interest within a period.

Specifically, if a point of social proof is the same location as this point of interest, then the

distance is 0 and the weight is 1, which can directly prove that the user has been to this point of interest within a period of time.

4.3 Connect Android Application to Smart Contracts

4.3.1 Web3

Generally speaking, Web3 is the next-generation worldwide network, a decentralized network [32].

Web 1.0

Understanding the previous versions of the Internet can help in understanding Web3.

For most users, in the first iteration of the web, they can only passively read web content.

The first phase of the Internet can be described in terms of how users interact with the web initially. For most users, they can only passively read web content in the first iteration of the web.

Web 2.0

Web2.0 is about the interactivity between users. At this stage, users can create and share content on the Internet. They participate in the network, but the disadvantage is that they provide too much personal data and information to the company that controls the platform.

What is Web3

Web3 is the future trend of the Internet, it is still under construction, so there is still a lack of a clear definition.

Generally speaking, web3 refers to the decentralized Internet. This means that no one party can control data or restrict access. Anyone can build and connect different decentralized applications on the decentralized network. Those participated in blockchain smart contract platforms, such as Ethereum, EOS are recognized as leaders of Web3 [33].

Web3.0 solves the main problem of web2.0, which is giant companies that control the platform can analyze and even sell according to the personal data of users.

There are some pillars to a Web3 project setup, such as Web3.js and Web3j. Web3j is the most advanced Ethereum JVM language library which has lightweight, highly modular, reactive and type-safe features. It connects to Ethereum nodes by using JSON-RPC of familiar standards like HTTP, WebSockets, and IPC, to interact with the Ethereum network.

4.3.2 Web3j

After researching related works and developing smart contracts and Android application, the next step is to connect the Android application to Ethereum by using web3j library.

Web3j is a Java and Android library for connecting smart contracts to nodes on the Ethereum network [32]. It has many vital features that can help implement this project as followings:

- Convert smart contracts to Java wrappers that can call functions of smart contracts by Java code.
- Support for Infura
- Android compatible

Usage

After using Truffle to compile smart contracts, corresponding JSON files will be generated. Then, developers can use command line tools of web3j to generate the Java wrappers [Github]:

```
$ cd /path/to/your/web3j/java/project

$ web3j truffle generate /path/to/<truffle-smart-contract-output>.json -o /path/
    to/src/main/java -p com.your.organisation.name
```

In addition, developers need to add “privateKeyRopsten”, “contractAddressRopsten” and “ropstenUrl” into Android code.

What is “privateKeyRopsten”? It is the private key of wallet. When developers installed MetaMask, they created an account and they can export the private key so that the Android application can connect to the MetaMask wallet.

What is “contractAddressRopsten”? It is the contract address of our network. When developers developed smart contracts and deployed it into Ethereum test network, they will get a contract address, where each transaction record will be displayed on this address [27]. This means that any transaction caused by the user’s interaction with smart contracts on the Android application can be viewed at this contract address.

Furthermore, what is “ropstenUrl”? It is a key applied to Infura. Typically, when developers would like to call smart contracts, they need to get the whole Ethereum blockchain while it may take a long time. One way to go around this limitation is to use a service like Infura. As mentioned in Section 2.6.5, Infura provides a service that allows developers to connect to a remote Ethereum node and execute transactions without having to worry about maintaining and synchronizing our local node [34].

Then, using an contract in Java code:

```
YourSmartContract contract = YourSmartContract.load("0x<address>|<ensName>", <
    web3j>, <credentials>, GAS_PRICE, GAS_LIMIT);
```

Hence, the Android application and Ethereum are successfully connected through web3j. Users interact with the Android application, such as set points of interest, verify points of social proof, will call the function of smart contracts.

Chapter 5

Software Testing and Evaluation

5.1 Testing the Smart Contracts

Testing is a vital part of the smart contract development and delivery process. It can effectively check whether the actual results meet the design expectations, and help identify errors and check for deficiencies. At the same time, high-quality, reusable testing also helps improve overall development efficiency.

Like traditional software, smart contract testing can also be divided into functional testing, non-functional testing and security testing. These tests will be introduced one by one below.

5.1.1 Functional Testing

Functional testing includes but is not limited to unit testing, integration testing, and user testing. Except for user tests, other tests can be implemented by developers or testers writing code.

The functional testing environment of smart contracts is divided into console and SDK code testing, which are suitable for different testing environments:

Console Testing

Truffle provides a command-line interactive interface for simple debugging by creating a contract on the console and entering call and query instructions. It is suitable for simple contracts.

First, the smart contract needs to be deployed. After the deployment is successful, the functions in the contract are called for testing. The following is an example of testing contract functions in the Truffle console:

```
> deploy verify-demo
contract address: 0xd931b41c70d2ff7b54eb9b2453072f9b1a4dc05c

> call verify-demo 0xd931b41c70d2ff7b54eb9b2453072f9b1a4dc05c checkInterestPoint
    (idHash)
transaction hash: 0
    xe569e5d8eae1b949a0ffe27a60f0b4c8bd839f108648f9b18879833c11e94ee4

--------------------------------------------------------------------------------
Output
function: checkInterestPoint(idHash)
return type: (bool)
return value: (true)
--------------------------------------------------------------------------------
```

The above example demonstrates how to deploy and debug a contract via the Truffle console. The console is simple in design and easy to use. However, when dealing with some complex scenarios, the single-line operation of the console is difficult to meet the requirements.

SDK Code Testing

In system testing, it is indeed to follow the classic AIR practical principles:

- Automatic: Testing should be executed automatically, which is also a prerequisite for continuous integration.
- Independent: The test cases remain independent of each other, and there is no mutual dependence and call.
- Repeatable: The test case must be reusable. It can be executed repeatedly across software and hardware environments.

To meet the above and even more test practice principles, the use of the Truffle console seems a little overwhelming, and the method of integrating the SDK and writing test code is more recommended. Although this method takes a long time and cost in the early stage, it can greatly reduce the repetitive workload in the later stage of the test and significantly improve the overall test efficiency.

For example, integrate Java web3j SDK, create a Java project, and develop and test code according to the following steps:

- Develop the smart contract: HelloWorld Contract.
- Compile smart contracts and use web3j to convert them to Java files.
- Based on the above files, call contract functions and interfaces, and write related test cases, such as ContractTest.
- Based on the gradle plugin provided by the Java project, run all test cases through the "./gradlew test" command.

The following is a test function in the core test code ContractTest:

```
function testAddInterestPoint() public {
    InterestPoint interestPoint = new InterestPoint();

    bytes32 studeidntId = 0
    x0000000000000000000000000000000000000000000000000000000000000001;

    bytes32 idHash = 0
    xa7834034bd059ecf00b0661f88f1e7242450bf1951c1e76803e80ce4182e2e9c;

    bytes32 expectedHash = 0
    xa7834034bd059ecf00b0661f88f1e7242450bf1951c1e76803e80ce4182e2e9c;

    smartDegree.addInterestPoint(id, idHash);

    Assert.equal(smartDegree.getIdHash(id), expectedHash, "hash should be equal"
    );
}
```

In the Java SDK, after any transaction is on the chain, users will get a receipt TransactionReceipt object, which contains the return status and error information (if the transaction is successful, the message is null), users can use this status to determine whether the transaction is normal, as shown below:

```
TransactionReceipt tr = interestPoint.set("InterestPoint").send();
Assert.assertEquals(tr.getMessage(), "0x0", tr.getStatus());
```

5.1.2 Non-Functional Testing

Non-functional testing mainly includes performance testing, capacity testing, and security testing. Since smart contracts run on the bottom nodes of Ethereum network, capacity testing and usability testing are more related to the bottom nodes. Therefore, for users, the focus of non-functional testing is on security testing.

Security Testing

Strict security tests are required before smart contracts published.

Security testing methods include: publicizing smart contracts and issuing rewards, hiring a dedicated smart contract security agency to detect and evaluate contracts, and using smart contract-specific tools for auditing. For smart contracts for individual developers or non-significant businesses, select free smart contract tools for testing is a better choice.

Taking VSCode as an example to demonstrate how to use the smart contract security plugin to check the security of the contract.

Open VSCode, search for Beosin-VaaS: ETH in its plugin library, and choose to install. Then, use this plugin to check the security of the selected contract. The following are the test results of the Verify.sol contract:

```
start compile!
compile over!
start deploy!
deploy over!
start execute function name: set(string)
execute function name : set(string) end!
No problem was found in the contract
---end---
```

5.2 Testing the Android Application

5.2.1 User Interface Testing

User interface (UI) testing can ensure that the functions of Android applications basically meet the requirements. One of the methods is to have testers simulate all user operations to verify that they are correct, while it is time-consuming and prone to omissions. Another way is to write UI tests and perform the test operations in an automated way.

Espresso

Espresso is a simple and easy to use Android UI testing framework, which are recommended by Google [35]. It is mainly composed of the following three basic parts:

- ViewMatcher
- ViewActions
- ViewAssertions

Examples of Espresso usage are as follows:

```
onView(ViewMatcher) // 1.Match View
    .perform(ViewAction) // 2.Perform View Action
        .check(ViewAssertion); // 3.Verify View
```

After writing all the UI test cases according to the above example, developers can execute all the test cases with the following command to get the testing results:

```
// Run Case under the src/test/ path
./gradlew testDebugUnitTest --continue

// Run Case under src/androidTest/ path
./gradlew connectedDebugAndroidTest --continue
```

5.3 Accomplishments

- A literature review was conducted to investigate the existing blockchain consensus and a series of principles that are critical to the project, as well as the technology for developing smart contracts.
- Realized the idea of social proofs based on the paper "*Proof of Witness Presence: Blockchain Consensus for Augmented Democracy in Smart Cities*" [2], including situation awareness and proving witnessing, but also made reasonable changes according to the actual needs, while meeting the use case of the working paper "*On cycling risk and discomfort: urban safety mapping and bike route recommendations*" [26].
- Deployment in the Ethereum network enables any citizen to access and verify the authenticity of the data, while also ensuring the immutability of the data.

5.4 Limitations

Although the final implementation has proven to work well in most cases, there are still some limitations.

Solidity does not support floating-point types

Solidity is an emerging language, but the current version still does not support floating-point types which means that when developers need to store some floating-point data into the block

of the blockchain [36], they need first to convert data to integer-type data, so as not to cause the loss of data details.

Google Maps cannot get the range of points of interest

In the analysis of Section 3.2.4, the designer of points of interest needs to set the range of points of interest. In fact, it is a more accurate approach to automatically obtain the size of the building where the point of interest is located, because some points of interest are tiny, while others are huge. For instance, a cafe on campus may have a range of only 20 meters, but the outdoor location like Hype Park maybe 500 meters in size. Thus, it is not appropriate to uniformly set the size of points of interest range to a fixed number, and it is also inaccurate to manually set the range size.

Therefore, if Google Maps can provide coverage of each geographic location, this project can accurately set the range of points of interest, so that it can more accurately assign weights and calculate whether it meets the requirements of social proofs, although Google Maps does not open this API for developers currently.

Not deployed on the Ethereum public chain

The smart contracts of this project are deployed on the Ethereum test network so that the test coins obtained are limited, which means that the gas and the number of transactions is limited. The system will have better performance if it is deployed on a developed public chain, but it will cost more.

Chapter 6

Conclusions and Future Work

In this section, a brief overview of the work carried out for the project, and suggestions for future work are aimed at improving and expanding the project.

6.1 Conclusions

The aim of the project was to develop smart contracts and deploy them on the blockchain that prove social claims on the smart phone, which was met successfully with the smart contracts on the Ethereum test network and a user-friendly Android application.

Moreover, six different scenarios were designed for this project, which can include almost all scenarios in real life. This allows citizens to use different physical evidence in different locations to implement social proof.

The final result basically achieves the expected effect, and some useful conclusions can be obtained in the developing process:

- It is feasible to achieve the decentralization of applications by developing smart contracts and deploying them to the Ethereum blockchain network.
- The realization of smart urbanization still requires strong hardware support and extensive and fast-speed network coverage.
- In the future, citizens will be in a decentralized network to avoid personal privacy data being manipulated by social media and giant Internet companies.

6.2 Future Work

Currently, the smart contracts are deployed on the Ethereum blockchain in this project, which has the following benefits:

- It is quite mature.
- It is easy to integrate.
- It has a mature community.

Nevertheless, there also has some factors that make Ethereum blockchain is not the most suitable choice for this project:

- Non-determinacy of consensus mechanism.
- Non-existence of roles.

The HyperLedger provides developers with a more suitable development environment than Ethereum. At this time, the HyperLedger Fabric is one of the most advanced and mature frameworks.

The Fabric has the concept of 'channel', which is a feature that make it unique. There is an isolation mechanism between different channels, which make Fabric consensus between different channels cannot interfere with each other. Nevertheless, for Ethereum, every consensus must be carried out on the entire network, which means the Fabric is inherently superior to Ethereum.

The developers also like its deterministic PBFT algorithm. Additionally, testing on the Docker containers is easy so that this project may consider deploying smart contracts on the consortium chain based on the HyperLedger Fabric.

References

[1] Sandi Gec, Dejan Lavbič, Marko Bajec, and Vlado Stankovski. Smart contracts for container based video conferencing services: Architecture and implementation. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 219–233. Springer, 2018.

[2] Evangelos Pournaras. Proof of witness presence: blockchain consensus for augmented democracy in smart cities. *Journal of Parallel and Distributed Computing*, 2020.

[3] Robert B Cialdini. *Influence: The new psychology of modern persuasion*. Morrow, 1984.

[4] José Rouillard. Contextual qr codes. In *2008 The Third International Multi-Conference on Computing in the Global Information Technology (iccgi 2008)*, pages 50–55. IEEE, 2008.

[5] Ioannis Karamitsos, Maria Papadaki, and Nedaa Baker Al Barghuthi. Design of the blockchain smart contract: A use case for real estate. *Journal of Information Security*, 9(3):177–190, 2018.

[6] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system.(2008), 2008.

[7] Marko Vidrih. What is a block in the blockchain. Available from: `https://medium.com/datadriveninvestor/what-is-a-block-in-the-blockchain-c7a420270373`, 2018.

[8] Yong Yuan and Fei-Yue Wang. Blockchain and cryptocurrencies: Model, techniques, and applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(9):1421–1428, 2018.

[9] Kejiao Li, Hui Li, Hanxu Hou, Kedan Li, and Yongle Chen. Proof of vote: A high-performance consensus protocol based on vote mechanism & consortium blockchain. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 466–473. IEEE, 2017.

[10] Yao Sun, Lei Zhang, Gang Feng, Bowen Yang, Bin Cao, and Muhammad Ali Imran. Blockchain-enabled wireless internet of things: Performance analysis and optimal communication node deployment. *IEEE Internet of Things Journal*, 6(3):5791–5802, 2019.

[11] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. A review on consensus algorithm of blockchain. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2567–2572. IEEE, 2017.

[12] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[13] Yunpeng Wang, Hui Zhao, Tao Li, Fan Zhang, and Zhiyong Li. Hybrid-chain: An innovative and efficient mixed blockchain architecture. In *2018 3rd International*

Conference on Electrical, Automation and Mechanical Engineering (EAME 2018). Atlantis Press, 2018.

[14] Soon-Gohn Kim. A study on the blockchain 2.0 ethereum platform analysis for dapp development. *The Journal of Korea Institute of Information, Electronics, and Communication Technology*, 11(6):718–723, 2018.

[15] Chris Grundy. Explained: How do ethereum transactions work? Available from: `https://thecoinoffering.com/learn/ethereum-transactions`, 2018.

[16] Damiano Di Francesco Maesa and Paolo Mori. Blockchain 3.0 applications survey. *Journal of Parallel and Distributed Computing*, 138:99–114, 2020.

[17] Nick Szabo. Smart contracts. *Unpublished manuscript*, 1994.

[18] Chris Dannen. Solidity programming. In *Introducing Ethereum and Solidity*, pages 69–88. Springer, 2017.

[19] Christian Reitwiessner and Gavin Wood. Solidity. 2015. *URL http://solidity. readthedocs. org*.

[20] Neetesh Mehrotra. An introduction to solidity, the language that runs ethereum. Available from: `https://www.opensourceforu.com/2019/08/an-introduction-to-solidity-the-language-that-runs-ethereum`, 2019.

[21] Elad Elrom. Ethereum wallets and smart contracts. In *The Blockchain Developer*, pages 173–212. Springer, 2019.

[22] Kedar Iyer and Chris Dannen. The ethereum development environment. In *Building games with ethereum smart contracts*, pages 19–36. Springer, 2018.

[23] Amari N Lewis and Amelia C Regan. Enabling paratransit and tnc services with blockchain based smart contracts. In *Science and Information Conference*, pages 471–481. Springer, 2020.

[24] Monika Di Angelo and Gernot Salzer. A survey of tools for analyzing ethereum smart contracts. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pages 69–78. IEEE, 2019.

[25] Wei-Meng Lee. Using the metamask chrome extension. In *Beginning Ethereum Smart Contracts Programming*, pages 93–126. Springer, 2019.

[26] David Castells-Graells, Christopher Salahub, and Evangelos Pournaras. On cycling risk and discomfort: urban safety mapping and bike route recommendations. *Computing*, pages 1–16, 2019.

[27] Evangelos Pournaras. Proofofsituationawareness. Available from: `https://github.com/epournaras/ProofOfSituationAwareness`, 2020.

[28] Simon Kim. Measuring ethereum's peer-to-peer network. 2017.

[29] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. Time-aware point-of-interest recommendation. In *Proceedings of the 36th international*

ACM SIGIR conference on Research and development in information retrieval, pages 363–372, 2013.

[30] Tan Jin Soon. Qr code. *Synthesis Journal*, 2008:59–78, 2008.

[31] ZXing Team. Zxing. *Obtido de Zxing-github: https://github. com/zxing/zxing*.

[32] Luke Hedger. Crossing over to web3 an introduction to decentralised development. Available from: `https://hackernoon.com/crossing-over-to-web3-an-introduction-to-decentralised-development-5eb09e95edb0`, 2017.

[33] Tran. What is web 3? Available from: `https://decrypt.co/resources/what-is-web-3`, 2019.

[34] Ondrej Bendo. Interacting with ethereum smart contracts from android. Available from: `https://blog.jayway.com/2017/08/23/interacting-with-ethereum-smart-contracts-from-android`, 2017.

[35] Denys Zelenchuk. *Android Espresso Revealed: Writing Automated UI Tests*. Apress, 2019.

[36] Zheng Yang and Hang Lei. Lolisa: Formal syntax and semantics for a subset of the solidity programming language. *arXiv preprint arXiv:1803.09885*, 2018.

Appendices

Appendix A

External Materials

The external materials used in the project include tools and frameworks, which are listed in the background research chapter.

A.1 Tools used in the project

- Blockchain
- Ethereum
- node.js
- npm
- web3j

A.2 Tools and Frameworks used in the Android component

- Android SDK
- Google Map API
- org.web3j:core:4.2.0-android
- com.blankj:utilcode:1.29.0
- com.github.clans:fab:1.6.4
- com.gyf.immersionbar:immersionbar:2.3.3
- site.gemus:openingstartanimation:1.0.0
- com.kongzue.dialog_v3:dialog:3.1.9
- com.shuhart.stepview:stepview:1.5.0
- com.tarun0.zxing-standalone:zxing-standalone:1.0.0

A.3 Tools and Frameworks used in the Smart Contract component

- Truffle
- MetaMask
- Infura

Appendix B

Source Code

All source code can be found at: https://github.com/epournaras/ProofOfSituationAwareness

Appendix C

Developer Setup Guide

C.1 Android Application Part

C.1.1 Prerequisites

- Android SDK installed.
- Google Map API Key.

C.2 Smart Contract Part

C.2.1 Prerequisites

- Node.js installed.
- web3j installed (version 4.2).

C.2.2 Installing Truflle

```
npm install -g truffle
```

C.2.3 Compiling the smart contract

```
truffle compile
```

C.2.4 Deploying the smart contract

```
# you could change the mnemonic and the address of ropsten in truffle-config.js
truffle migrate --network ropsten # --reset optional
```

C.2.5 Generating Java file

```
web3j truffle generate [LOCATION]/verify-sol/build/contracts/Verify.json -o [
    LOCATION]/verify-demo/app/src/main/java/ -p ac.hurley.verifydemo.contract
```